Løsning på småoppgavene i boka

Dette dokumentet er kun til bruk i tilknytning til følgende bok: Else Lervik og Mildrid Ljosland: "Programmering i C++. En innføring i strukturert og objektorientert programmering." Stiftelsen TISIP og Gyldendal Akademisk 2003. ISBN 82-05-30733-4.

Kapittel 1 - 1

Oppgave 2

I Windows: Bruk Utforsker og velg Fil/Ny/Mappe. Endre navnet til Prog. Gå til Prog og velg Fil/Ny/Mappe på nytt. Endre navnet til Kap01. Fortsett til du har en underkatalog for hvert kapittel.

Kapittel 1 - 3

Oppgave 1

Definisjoner:

```
const double strykGrense = 10.0;
double res1;
double res2;
double res3;
double total;
```

Aktive setninger:

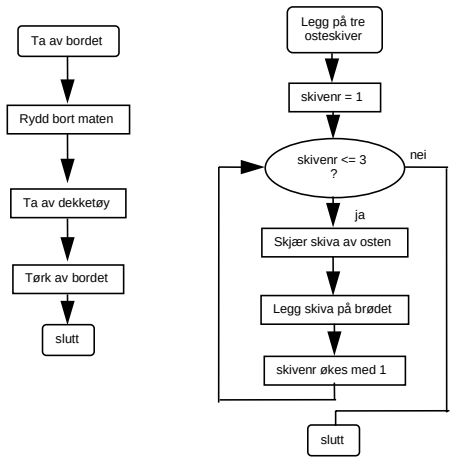
```
1 cout << "Skriv tre resultater mellom 0.0 og 20.0: ";
2 cin >> res1 >> res2 >> res3;
3 total = res1 + res2 + res3;
4 total = total / 3.0;
5 cout << "Totalresultatet blir " << total << endl;
6 if (total < strykGrense) {
7   cout << "Stryk!" << endl;
8 return 0;
```

Setning 6 er en valgsetning, alle de andre er sekvensielle setninger. Setning 1 og 5 er skrivesetninger og setning 2 er en lesesetning. Setning 3 og 4 er tilordningssetninger og setning 8 er en retursetning.

Oppgave 2

```
//-----------------------------------------------------
//
// Program for å finne totalresultatet fra fire
// delresultater
//
#include <iostream>
using namespace std;
int main()
{
  const double strykGrense = 10.0;
  double res1;
  double res2;
  double res3;
  double res4;

  cout << "Skriv fire resultater mellom 0.0 og 20.0:";
  cin >> res1 >> res2 >> res3 >> res4;

  double total;     // Totalkarakteren
  total = res1 + res2 + res3 + res4;
  total = total / 4.0;

  cout << "Totalkarakteren blir " << total << endl;
  if (total < strykGrense) {
    cout << "Stryk!" << endl;
    }
  return 0;
} // main
```

Kapittel 1 – 5



Kapittel 2-1

Oppgave 1

```
#include <iostream>
using namespace std;

int main()
{
  double grunnlinje;
  double hoyde;

  cout <<
    "Dette programmet beregner arealet til en trekant."
    << endl;
  cout << "Oppgi grunnlinje: ";
  cin >> grunnlinje;
  cout << "Oppgi høyde: ";
  cin >> hoyde;

  double areal;
  areal = grunnlinje * hoyde * 0.5;
  cout << "Arealet til en trekant med grunnlinje "
    << grunnlinje << " og høyde "
    << hoyde << " er " << areal << endl;
  return 0;
} // main
```

Kapittel 2-2

Oppgave 1

Følgende ord er lovlige navn i C++:

```
KarisBok
NUMMER
evas_sykkel
_Tall34
```

Oppgave 2

a) Programmet inneholder 9 setninger.

b) Følgende ferdigdefinerte ord brukes:

```
using
namespace
std
int
main
double
const
cin
```

```
cout
endl
return
```

Strengt tatt er ikke **main** ferdigdefinert, vi definerer det her, men for at programmet skal virke må vi bruke ordet **main**.

c) Følgende egendefinerte ord brukes:

```
kCal
kJ
omregningsfaktor
```

Kapittel 2-3

Oppgave 1

a) **double**

b) **double**

c) **int** eller **long int** (kompilatoravhengig)

d) **char**

e) **int** eller **long int** (kompilatoravhengig)

Oppgave 2

To av definisjonene er lovlige:

```
const double min  = 5.67;
const char   tegn = 'Æ';
```

Oppgave 3

Navnløse konstanter:

```
4.2
"Antall kCal: "
"Antall kJ blir "
0
```

Navngitt konstant:

```
omregningsfaktor
```

Variabler:

```
kCal
kJ
```

Oppgave 4

3.5/5.6 blir 0.625
6/3 blir 2
7%3 blir 1
5%4 blir 1
10/4 blir 2

Oppgave 5

Vi må regne om fra 2-tallsystemet til 10-tallsystemet:

tegn1: $0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 49$
tegn2: $1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 68$
tegn3: $1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 65$

Men 49 er ASCII-verdien til '1', 68 til 'D' og 65 til 'A'.

Resultatet av **cout**-setningen blir da utskriften **1DA**

Oppgave 6

De tre tallverdiene blir nå skrevet ut: **496865** (ikke mellomrom). For å få mellomrom mellom tallene, må **cout**-setningen se slik ut:

```
cout << tegn1 << ' ' << tegn2 << ' ' << tegn3 << endl;
```

Oppgave 7

Det oktale tallsystemet er et annet navn på 8-tallsystemet. Sifrene 0-7 benyttes. 42 i 8-tallsystemet blir $4 \cdot 8^1 + 2 \cdot 8^0 = 32 + 2 = 34$ i 10-tallsystemet.

Kapittel 2-5

Oppgave 1

Følgende setninger er lovlige:

```
b = b + c;
d = d;
```

Oppgave 2

I rekkefølge: 25, 25, 7, 3, 5.8303..., 5, 6
(Husk heltallsdivisjon når to heltall divideres med hverandre!)

Oppgave 3

a blir 10, **b** blir 0, **d** blir 0. **c, p** og **q** endrer ikke verdi.

Oppgave 4 og 5

```
#include <iostream>
using namespace std;
int main()
{
  double a = 10.0;
  double b = 5.0;
  double c = 3.5;
  double d = 2.2;
  double x = 2.8;
  double y = 3.2;
  double z = 4.3;
  double sa = b * b - 4.0 * a * c;
  double sb = x * x + y * y + z * z;
  double sc = x * x * x;
  double sd = (a - b) * (a + b);
  double se = (a + b) / (c + d);
  cout << sa << ' ' << sb << ' ' << sc << ' ' <<
    sd << ' ' << se << endl;
  return 0;
} // main
```

Kjøring av programmet gir følgende utskrift:

```
-115 36.57 21.952 75 2.63158
```

Oppgave 6

Resultatet avhenger av tallområdet for **int** og **long int**. Anta at 60000000000 er utenfor tallområdet til **int**, men innenfor tallområdet til **long int**. Da vil **a** kunne være hva som helst, **b** blir 60000000000. Husk at tilordning skjer ved at høyre side regnes ut først, deretter foretas eventuelt omforming til den datatypen som variabelen på venstre side tilhører. Ved beregning av **a** vil høyre side regnes ut som et uttrykk av typen **int**, men ettersom resultatet av beregningen ligger utenfor tallområdet, vil det kunne være hva som helst. Etter at uttrykket er beregnet, omformes resultatet til **long int**. Ved beregning av **b**, vil høyre side regnes ut som et uttrykk av typen **long int**, ettersom operandene er av denne typen, og resultatet blir korrekt.

Kapittel 2-6

Oppgave 1

```
int tall1 = 1;
int tall2 = 2;
int tall3 = 3;
int hjelp;
hjelp = tall1;
tall1 = tall2;
tall2 = tall3;
tall3 = hjelp;
cout << tall1 << " " << tall2 << " " << tall3 << endl;
```

gir følgende utskrift:

```
2 3 1
```

Kapittel 2-7

Oppgave 1

Det trengs 5 **char**-variabler:

```
int  tall;
char t1;
char t2;
char t3;
char t4;
char t5;
cin >> tall >> t1 >> t2 >> t3 >> t4 >> t5;
```

Ved innlesing omformes tegnet '3' til binær form (datatypen **int**). De resterende tegnene omformes ikke. Formateringen sørger imidlertid for at blanke og linjeskift hoppes over.

Oppgave 2

```
34ABC
SLUTT
```

Kapittel 3-1

Oppgave 1

Vi har navngitt konstantene 'T' og 'S'. Da er det viktig at vi bruker navnene. En som skal vedlikeholde programmet vil kanskje ønske å endre f.eks. 'T' til 't'. Ettersom vi har en navngitt konstant, vil vedkommende nøye seg med å forandre verdien til den, og høyst sannsynlig overse de stedene der konstantnavnet ikke er brukt.

Oppgave 2

Innlesing av grunnlinje og høyde blir nå felles for de to figurene. Dette kan derfor flyttes ut i ytterste blokk. **if**-setningen blir da som følger:

```
if (valg == trekant) areal = grunnlinje * hoyde * 0.5;
else areal = grunnlinje * hoyde;
```

Eller:

```
areal = grunnlinje * hoyde;
if (valg == trekant) areal = areal *0.5;
```

Oppgave 3

Datakrav		
Inndata	**kCal**	datatypen **double**. KiloKalori
Utdata	**kJ**	datatypen **double**. KiloJoule
Interne data	Ingen.	
Konstanter	**omregningsfaktor**	datatypen **double**. Lik 4,2.

Kapittel 3-2

Oppgave 1

Programmet inneholder tre blokker:

```
int main()
{                                   start på blokk 1
  const ...
  ..
  if (valg == trekant) {            start på blokk 2
    ...
    } // trekantberegning           slutt på blokk 2

  else { // sirkelberegning         start på blokk 3
    ....
    } // sirkelberegning            slutt på blokk 3
  ...
} // main                           slutt på blokk 1
```

Oppgave 2

Den lengst til høyre.

Oppgave 3

I første del av **if**-setningen økes **a** med verdien til **b**. I andre del av **if**-setningen redefineres **a**: **int a = a + b;** Også **a**'en på høyre side av tilordningstegnet er den nye **a**'en. Den har ingen bestemt verdi. Utskrift av den kan gi hva som helst. Til slutt i programbiten skrives verdien til uttrykket **a - b** ut.

Tilfelle 1: Inndata er 4 og 5. Da vil **else**-blokka bli utført. Utskriften blir:

```
7164 5
-1
```

Det første tallet er den **a**'en som er definert inne i blokka, og den kan være hva som helst. Siste linje er verdien til uttrykket **a - b**, og nå er vi i ytre blokk, og det er den **a** som ble lest inn som gjelder.

Tilfelle 2: Inndata er 20 og10. Første del av **if**-setningen utføres, og utskriften blir (**a** i ytre blokk er øket med 10):

```
30 10
20
```

Kapittel 3-3

Oppgave 1

```
if (antall > 20) kode = 'M';
else kode = 'F';

if (temp > 25) cout << "varmt";
else {
  if (temp > 14) cout << "passe";
  else cout << "kjølig";
  }
```

Oppgave 2

Teksten "Liten!" skrives ut hvis **stoerrelse** er mindre eller lik 36. Setningene bør skrives slik:

```
if (Stoerrelse <= 36) cout << "Liten!";
```

Oppgave 3

Utskriften blir:

```
Hei!Godmorgen!
```

Oppgave 4

```
if (a > b) {
  b = a + b;
  if (b > c) cout << "Hei!";
  }
else cout << "Hallo!";
cout << "Godmorgen!";
```

Kapittel 3-4

Oppgave 1

a) Venstre side blir 15/4 + 3 lik 3 + 3 lik 6, som er mindre enn 10. Uttrykket er sant.

b) 4 > 3 er sant. Men "ikke"-operatoren snur verdien. Svaret blir dermed at uttrykket !(4 < 3) er usant.

c) Første del er sann, andre del er usann. "og"-operatoren krever at begge delene må være sanne. Resultat: Uttrykket er usant.

d) Første del er sann, andre del er usann. For "eller"-operatoren er det nok at en av delene er sann. Resultat: Uttrykket er sant.

e) Verdien til et tilordningsuttrykk er verdien til variabelen på venstre side etter tilordningen. Det vil si 20. Den logiske verdien er sann, ettersom alle verdier forskjellig fra 0 er sanne.

f) Aritmetisk verdi 0. Logisk verdi usann.

g) Aritmetisk verdi 0. Logisk verdi usann.

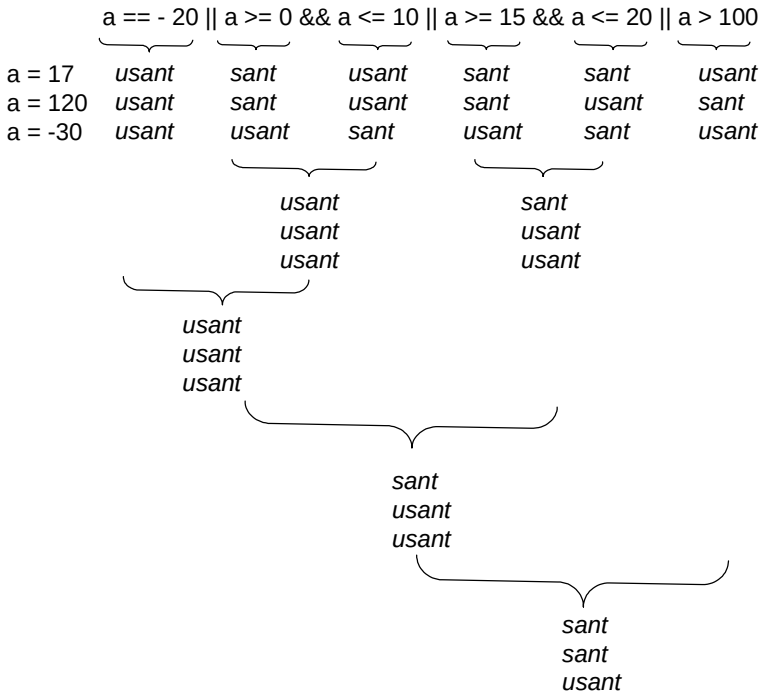
h) Aritmetisk verdi -20. Logisk verdi sann.

Oppgave 2

```
a)  antallElever > 20 && antallElever < 30

b)  loddNr == 3 || loddNr == 18 || loddNr == 25

c)  svar == 'j' || svar == 'J'

d)  temp < 15 || temp > 25

e)  sum > 0 && sum < 10 || sum > 100

f)  tegn >= 'A' && tegn <= 'Z' || tegn >= 'a' && tegn <= 'z' || tegn
== 'æ' || tegn == 'ø' || tegn == 'å' || tegn == 'Æ' || tegn == 'Ø'
|| tegn == 'Å'

g)  tegn >= '0' && tegn <= '9'
```

Oppgave 3

a = 17 og a = 120 gir at verdien til uttrykket er sant, mens a = -30 gir verdien usann.



Kapittel 3-5

Oppgave 1

```
#include <iostream>
using namespace std;
int main()
{
  const char   sirkel = 'S';
  const char   trekant = 'T';
  const char   rektangel = 'R';
  const char   kvadrat = 'K';
  const char   sirkelLiten = 's';
  const char   trekantLiten = 't';
  const char   rektangelLiten = 'r';
  const char   kvadratLiten = 'k';

  const double pi = 3.141592;

  char valg;
  cout << "Trekant, sirkel, kvadrat eller rektangel (" <<
```

```
    trekant << "/" << sirkel << "/" << kvadrat <<"/" <<
    rektangel << ")? ";
  cin >> valg;

  // Omformer til stor bokstav dersom brukeren
  // har skrevet inn liten:
  if (valg == rektangelLiten) valg = rektangel;
  else if (valg == sirkelLiten) valg = sirkel;
  else if (valg == trekantLiten) valg = trekant;
  else if (valg == kvadratLiten) valg = kvadrat;

  // Sjekker om valget er gyldig.
  bool gyldigValg = false;
  if (valg == sirkel || valg == kvadrat || valg == rektangel ||
      valg == trekant) {
    gyldigValg = true;
  }

  if (gyldigValg) {
    double areal;
    if (valg == sirkel) {
      double radius;
      cout << "Radius: ";
      cin >> radius;
      areal = pi * radius * radius;
      } // sirkel-beregning

    else if (valg == kvadrat) {
      double side;
      cout << "Lengden av en side: ";
      cin >> side;
      areal = side * side;
      } // kvadratberegning

    else { // enten rektangel eller trekant
      double grunnlinje;
      double hoyde;
      cout << "Grunnlinje og høyde: ";
      cin >> grunnlinje >> hoyde;
      if (valg == trekant) areal = grunnlinje * hoyde * 0.5;
      else areal = grunnlinje * hoyde;
      } // enten rektangel eller trekant
    cout << "Arealet blir " << areal << endl;
    }

  else cout << "Ugyldig valg." << endl;

  return 0;
} // main
```

Oppgave 2

Tilordningstegn er brukt i stedet for dobbelt likhetstegn. Det vil si at første uttrykk alltid er sant. Selv om vi retter opp denne feilen, så har vi flere:

if-setningene er plassert inni hverandre. Programmet kommer bare til andre **if**-setning dersom **spraak** = = **'E'**, og da er iallfall ikke **spraak** lik **'T'**. Flervalgsetningen må skrives slik (eventuelt uten **{ }** ettersom bare en setning skal utføres for hvert alternativ):

```
if (spraak == 'E') {
  cout << "Good morning!";
  }
else if (spraak == 'T') {
  cout << "Guten Morgen!";
  }
else if (spraak == 'S') {
  cout << "God morron!";
  }
else {
  cout << "God morgen!";
  }
```

Kapittel 3-6

Oppgave 1

```
if (poeng < min || poeng > maks) {
    cout << "Ugyldig poengsum. Må være i intervallet ["
         << min << ", " << maks << "]";
  }
else if (poeng < grenseE) {
  cout << "Karakteren blir F: Ikke bestått.";
  }
else if (poeng < grenseD) {
  cout <<
  "Karakteren blir E: Tilfredsstiller minimumskravene.";
  }
else if (poeng < grenseC) {
  cout << "Karakteren blir D: Under gjennomsnittet.";
  }
else if (poeng < grenseB) {
  cout << "Karakteren blir C: Gjennomsnittlig prestasjon.";
  }
else if (poeng < grenseA) {
  cout << "Karakteren blir B: Meget god prestasjon.";
  }
else cout << "Karakteren blir A: Fremragende prestasjon!";
 cout << endl;
```

Kapittel 4 - 1

Oppgave 1

```
//------------------------------------------------------
//
// Funksjon som beregner sum, differanse eller produkt
//
int beregnSvar(
  int  tall1,           // Inn
  int  tall2,           // Inn
  char regneart)        // Inn
{
  int svar;
  if (regneart == '+') svar = tall1 + tall2;
  else if (regneart == '-') svar = tall1 - tall2;
  else svar = tall1 * tall2;
  return svar;
} // beregnSvar
```

Kapittel 4 - 2

Oppgave 1

Alle steder det står **int**, må forandres til **double**. Ellers ingen forandring.

Oppgave 2

Oppgave a)

Kompileringsfeil. Kompilatoren finner ikke en prototyp som passer til funksjonskallet.

Oppgave b)

```
svar = beregnSvar(regneart, tall1, tall2);
```

regneart blir tolket som et tall (ASCII-verdien til tegnet). '+' blir til tallet 43, '-' til 45. **tall2** blir tolket som et tegn. Tegnet blir '+' hvis **tall2** er 43, da blir addisjon utført. I alle andre tilfeller blir subtaksjon utført.

```
svar = beregnSvar(tall2, tall1, regneart);
```

regner ut **tall2** +/- **tall1** i stedet for **tall1** +/- **tall2**.

Kapittel 4 - 3

Oppgave 1

```
//------------------------------------------------------
//
// Et program som tester funksjonen maks
//
#include <iostream>
using namespace std;
double  maks(double tall1, double tall2);

int main()
{
  double tallA;
  double tallB;
  cout << "Skriv to tall: ";
  cin >> tallA >> tallB;
  cout << "Det største er " << maks(tallA, tallB) << endl;
  return 0;
} // main

//------------------------------------------------------
//
// Funksjon som bestemmer det største av to tall
//
double maks(
  double tall1,            // Inn
  double tall2)            // Inn
{
  if (tall1 > tall2) return tall1;
  else return tall2;
} // maks
```

Oppgave 2

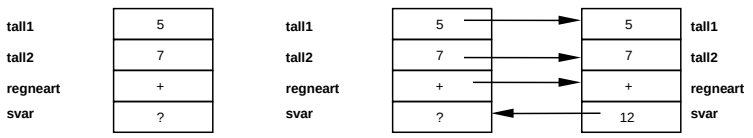
```
//------------------------------------------------------
//
// Program som tester funksjonen tilStor
//
#include <iostream>
#include <cctype>
#include <locale>
using namespace std;
char tilStor(char tegn);

int main()
{
  if (tilStor('a') == 'A') cout << "OK1" << endl;
  if (tilStor('æ') == 'Æ') cout << "OK2" << endl;
  if (tilStor('ø') == 'Ø') cout << "OK3" << endl;
  if (tilStor('å') == 'Å') cout << "OK4" << endl;
  if (tilStor('5') == '5') cout << "OK5" << endl;
  return 0;
} // main
```

```
//------------------------------------------------------
//
// Funksjon for å omforme en liten bokstav til en stor
//
char tilStor(
  char tegn)       // Inn
{
  if (tegn == 'æ') return 'Æ';
  else if (tegn == 'ø') return 'Ø';
  else if (tegn == 'å') return 'Å';
  else return toupper(tegn);
} // tilStor
```

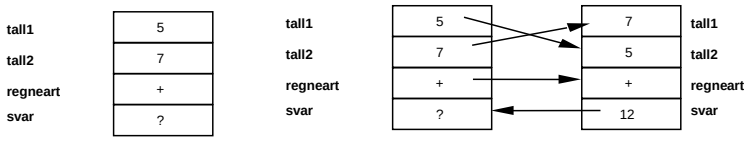
Kommentar: Denne funksjonen fungerer i et homogent miljø, dvs. hvis vi kjører i et rent Windows-miljø eller et rent UNIX-miljø. Men hvis vi kjører en blanding av Windows og DOS, slik som tilfellet er hvis vi lager en konsoll-applikasjon i Visual C++, får vi problemer med tegnsettet. Derfor har vi ikke prøvd å lese inn eller skrive ut de særnorske tegnene, for da ville ikke funksjonen ha gitt riktig resultat.

Kapittel 4 - 5

Oppgave 1



A: Svar = beregnSvar(tall1, tall2, regneart)



B: svar = beregnSvar(tall2, tall1, regneart)

Oppgave 2

Hvis en lokal variabel har samme navn som et formelt argument, får vi kompileriringsfeil. Å ha en lokal variabel med samme navn som et aktuelt argument, går greitt.

Kapittel 5-1

Oppgave 1

Initiering: **teller = 0;**
Løkkebetingelse: **teller < 5**
Innhold: **cout << endl;**
Oppdatering av betingelsen: **teller = teller +1;**

Oppgave 2

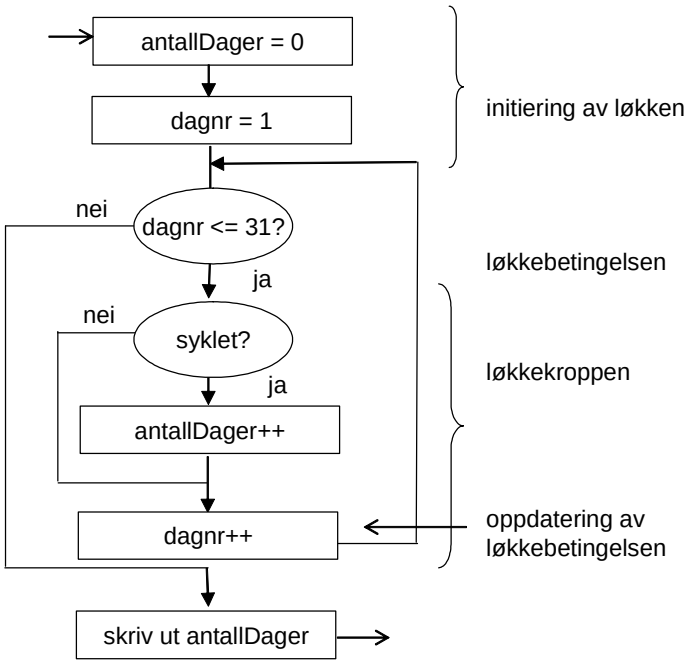
```
// Løkka utføres 0 ganger:
teller = 0;
while (teller < 0) {
  cout << teller << endl;
  teller = teller + 1;
  }

// Løkka utføres 1 gang:
teller = 0;
while (teller < 1) {
  cout << teller << endl;
  teller = teller + 1;
  }

// Løkka utføres uendelig mange ganger:
teller = 0;
while (teller < 1) cout << teller << endl;
```

Oppgave 3

Oppgave a:

Initiering: Antall dager Anders har syklet settes til 0. Dagnr til 1.
Løkkebetingelse: Flere dager igjen i mai? (Dagnr <= 31) Ev. kan vi se bare på skoledager. Antallet er uansett kjent før løkka.
Innhold: Finn ut om Anders syklet den dagen. Øk i så fall antall sykkel-

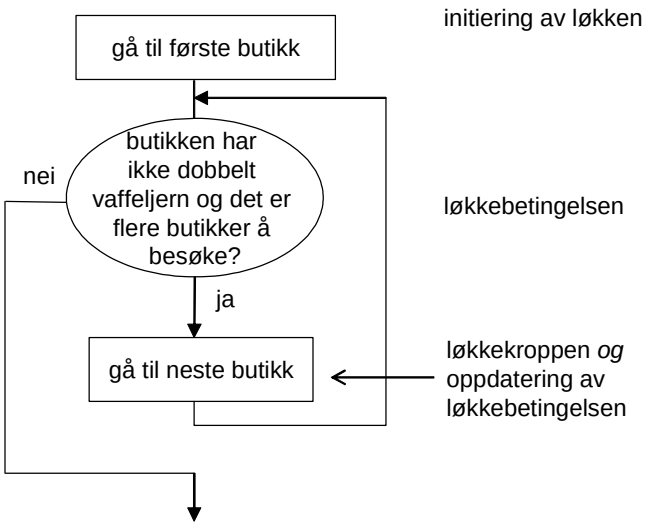
dager med 1.
Oppdatering: Dagnr økes med 1.



Oppgave b:

Vi antar at det er minst én butikk å besøke. Vi ser bort i fra at Eva går tilbake til tidligere besøkte butikker.

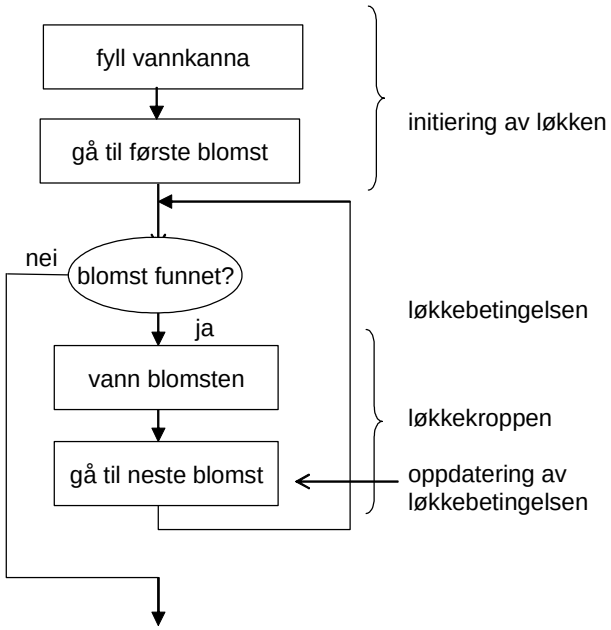
Initiering: Se etter den første butikken.
Løkkebetingelse: Vaffeljern ikke funnet og flere butikker å lete i.
Innhold og oppdatering: Gå til neste butikk.



Oppgave c:

Vi antar at ei kanne vann er nok til å vanne alle blomstene.

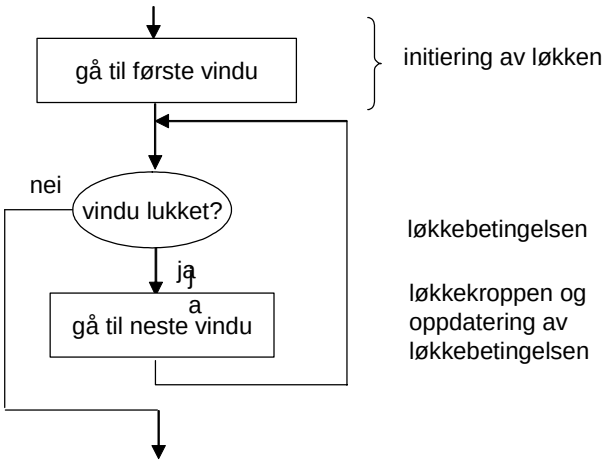
Initiering: Fyll vannkanna og gå til den første blomsten.
Løkkebetingelse: Flere blomster.
Innhold: Vann blomsten.
Oppdatering: Gå til neste blomst.



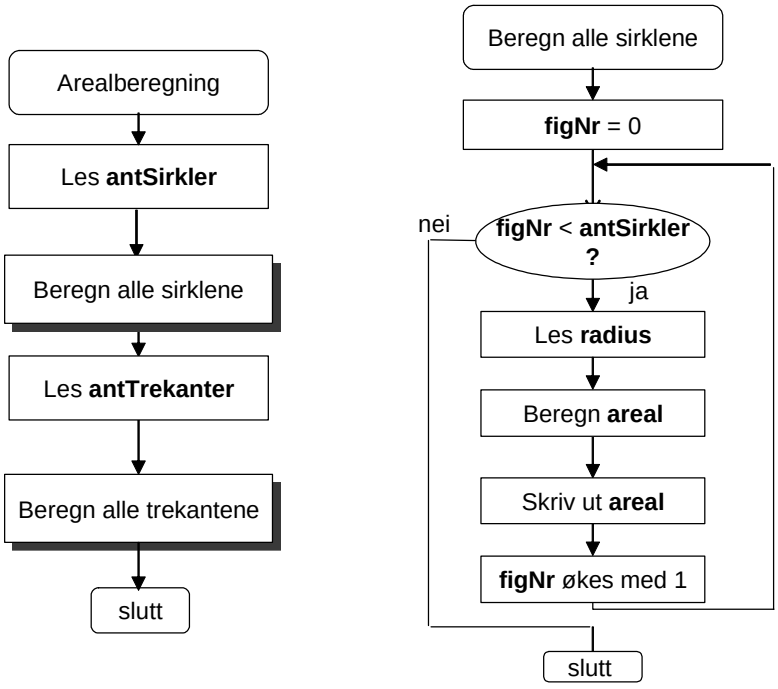
Oppgave d:

Vi antar at Åse klatrer inn i det første åpne vinduet hun finner.

Initiering: Gå til det første vinduet..
Løkkebetingelse: Er det lukket?
Innhold og oppdatering: Gå til neste vindu.


gå til første vindu
initiering av løkken
nei
vindu lukket?
løkkebetingelsen
ja
ja
gå til neste vindu
løkkekroppen og
oppdatering av
løkkebetingelsen

Oppgave 4


Arealberegning
Les **antSirkler**
Beregn alle sirklene
Les **antTrekanter**
Beregn alle trekantene
slutt
Beregn alle sirklene
figNr = 0
nei
figNr < **antSirkler** ?
ja
Les **radius**
Beregn **areal**
Skriv ut **areal**
figNr økes med 1
slutt

Detaljeringen av "Beregn alle trekantene" blir helt tilsvarende.

Lista over datakrav forandres. I stedet for inndataene **flere** og **valg** får vi de to heltallsvariablene **antTrekanter** og **antSirkler**. Den interne variabelen **figNr** holder orden på hvilken trekant eller sirkel programmet holder på med til enhver tid. Her er programmet:

```
#include <iostream>
#include <cctype>
using namespace std;

int main()
{
  const double pi = 3.141592;

  // Først alle sirklene
  int antSirkler;
  cout << "Hvor mange sirkler? ";
  cin >> antSirkler;
  int figNr = 0;
  while (figNr < antSirkler) {   // sirkler
    double radius;
    cout << "Skriv radius : ";
    cin >> radius;
    double areal = pi * radius * radius;
    cout << "Arealet blir " << areal << endl;
    figNr = figNr + 1;
    }

  // Deretter alle trekantene
  int antTrekanter;
  cout << "Hvor mange trekanter? ";
  cin >> antTrekanter;

  figNr = 0;
  while (figNr < antTrekanter) {
    double grunnlinje;
    double hoyde;
    cout << "Skriv grunnlinje og høyde : ";
    cin >> grunnlinje >> hoyde;
    double areal = grunnlinje * hoyde * 0.5;
    cout << "Arealet blir " << areal << endl;
    figNr = figNr + 1;
    }

  return 0;
} // main
```

Kapittel 5-2

Oppgave 1

Programmet må ha en verdi på **tall** første gangen løkkebetingelsen utføres.

Oppgave 2

Løkka utføres 0 ganger. **sum** har verdien 0.

Oppgave 3

Lista over inndata utvides med **antallTall**. Programmet må videre vedlikeholde en "teller" (f.eks. **tallNr**) som teller antall tall som leses inn. Algoritmen blir som til høyre i løsningen på oppgave 4, kap. 5-1. I stedet for antall sirkler skal nå et visst antall tall behandles.

```
#include <iostream>
using namespace std;
int main()
{
  int antallTall;
  cout << "Hvor mange tall skal summeres? ";
  cin >> antallTall;

  int sum = 0;
  int tallNr = 0;
  while (tallNr < antallTall) {
    int tall;
    cout << "Skriv tall nr " << (tallNr +1) << ": ";
    cin >> tall;
    sum = sum + tall;
    tallNr = tallNr + 1;
    }
  cout << "Summen er " << sum << endl;
  return 0;
} // main
```

Oppgave 4

Korrekt løsning (krever at n >= 0):

```
#include <iostream>
using namespace std;
int main()
{
```

```
  int x;    Nødvendig å definere X og N
  int n;
  cout << "Skriv grunntallet x og eksponenten n: ";
  cin >> x >> n;       Ikke komma
  long int svar = 1;   NB! Ikke null
  int teller = 0;      Må starte på 0, hvis betingelsen
  while (teller < n) {     skal være (teller < n)
    svar = svar * x;   Gange med x, ikke n
    teller = teller + 1;   Øke teller, ikke n
    }
  cout << "x opphøyd i n er " << svar << endl;
  return 0;
} // main
```

Kapittel 5-3

To "tellere" trengs, en som teller opp alle tallene som leses inn, og en som teller bare de som inngår i summen. Dessuten trengs det en intern variabel, som beregner summen av alle tallene som leses inn. Pass på at avslutningstallet (0 eller negativt) ikke kommer med, verken når antall tall skal telles, eller i summen.

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Skriv tallene som skal summeres." << endl;
  cout << "Avslutt med 0 eller negativt tall" << endl;

  int forrigeTall = -1; // en verdi som ikke forekommer blant
                        // tallene som skal summeres

  int sumAlle = 0;
  int antallTallTotalt = 0;
  int antallTallSummert = 0;

  int sum = 0;
  int tall;
  cin >> tall;
  while (tall > 0) {
    sumAlle = sumAlle + tall;
    antallTallTotalt = antallTallTotalt + 1;
    if (tall != forrigeTall) {
      sum = sum + tall;
      antallTallSummert = antallTallSummert + 1;
      }
    forrigeTall = tall;
    cin >> tall;
    }
  cout << "Summen er " << sum << endl;

  if (antallTallSummert > 0) {
    double snitt = (double) sum / (double) antallTallSummert;
```

```
    cout << "Antall tall summert er " << antallTallSummert <<
            "; og gjennomsnittet av disse er " << snitt << endl;
  } else cout << "Ingen tall er summert." << endl;

  if (antallTallTotalt > 0) {
    double snitt =
          (double) sumAlle / (double) antallTallTotalt;
    cout << "Totalt antall tall lest inn er " <<
      antallTallTotalt <<  "; og gjennomsnittet av disse er "
      << snitt << endl;
  } else cout << "Ingen tall er lest inn." << endl;

  return 0;
} // main
```

Kapittel 5-4

Oppgave 1

```
int dag;
for (dag = 1; dag <= antDager; dag++) {
  cout << "Dag nr " << dag << ": Fint vær!" << endl;
  }
```

Oppgave 2

Vi bruker **for**-løkker dersom vi vet antall løkkegjennomløp på forhånd.

a) Antall dager i mai er kjent. Her kan vi altså like gjerne bruke ei **for**-løkke.

b) Dersom Eva bor på et mindre sted er det sannsynlig at hun vet eksakt antall elektrobutikker. Så spørs det om hun vil sjekke alle butikkene før hun bestemmer seg. Da bruker vi i tilfelle ei **for**-løkke. Hvis hun ikke er sikker på at hun vil besøke alle butikkene, bruker vi ei **while**-løkke og avslutter enten når alle butikkene er besøkt eller huin har funnet et vaffeljern hun vil kjøpe.

c) For å kunne bruke ei **for**-løkke må Tore vite hvor mange blomster han har i stua.

d) Hvis huset er lite, vet antakelig Åse hvor mange vinduer det har. Imidlertid antar vi at hun benytter seg av det første åpne vinduet hun finner. Vi bruker ei **while**-løkke.

Oppgave 3

antall++ betyr at innholdet i variabelen **antall** økes med 1. Tilsvarende betyr **antall--** at innholdet minskes med 1.

antall++; tilsvarer **antall = antall + 1;**
antall--; tilsvarer **antall = antall - 1;**

Oppgave 4

Tabellen viser innholdet i variablene etterhvert som setningene utføres:

setning	k	a	b	c	d	e
double k = 3;	3					
k++;	4					
int a = 3;		3	?	?	?	?
int b = 4;			4			
a++;		4				
int c = a % b;				0		
int d = c % b;					0	
d--;					-1	
int e = a + b * 2 - b % 3 / 5;						12

Siste uttrykk beregnes som om det sto: a + (b * 2) - ((b % 3) / 5)

Utskriften blir som følger:

```
4 4 0 -1 12
```

Kapittel 5-5

Oppgave 1

Programbiten leser inn et visst antall tegn, og antall ganger disse skal skrives ut på skjermen. Dersom tegnet er lykketegnet (navngitt konstant eller lest inn tidligere), skrives tegnet ut dobbelt så mange ganger som oppgitt.

Forslag til korrekt program:

```
#include <iostream>
using namespace std;
int main() {
  const char lykkeTegn = '*';
  int antall;
```

```
  cout << "Antall tegn: ";
  cin >> antall;
  int teller;
  for (teller = 0; teller < antall; teller++) {    semikolon
    cout << "Skriv tegnet og antall ganger det skal "
      << "skrives ut: ";
    char tegn;   semikolon
    int antallGanger; semikolon
    cin >> tegn >> antallGanger;
    if (tegn == lykkeTegn) {
      antallGanger = antallGanger * 2;
      }    manglet slutt-klamme
    Neste for-løkke er inni for-løkken med teller som løkke-
    variabel. Må derfor lage en ny løkkevariabel.
    int teller2;
    for (teller2 = 0; teller2 < antallGanger; teller2++) {
      cout << tegn;
      }
    Ved å flytte neste setning en linje opp får vi linjeskift etter
    hver linje med tegn
    cout << endl;
    }
  return 0;
} // main
```

Oppgave 2

Programmet kan se slik ut:

```
#include <iostream>
using namespace std;
int main()
{
  int antall;
  cout << "Antall tall: ";
  cin >> antall;
  cout << "Adder eller multipliser? Skriv + eller *: ";
  char operatoren;
  cin >> operatoren;
  cout << "Skriv tallene: ";
  int svar;
  int teller;
  if (operatoren == '+') {
    svar = 0;      initierer til 0 ved addisjon
    for (teller = 0; teller < antall; teller++) {
      int tall;
      cin >> tall;
      svar = svar + tall;
      }
    }
    else {
    svar = 1;      initierer til 1 ved multiplikasjon
    for (teller = 0; teller < antall; teller++) {
      int tall;
```

```
        cin >> tall;
        svar = svar * tall;
        }
      }
    cout << "Svaret blir " << svar << endl;
    return 0;
  } // main
```

if-setningen hører hjemme utenfor løkka på grunn av at programmet ikke skal velge hvorvidt det skal addere eller multiplisere for *hvert* tall som leses inn. Ved store mengder tall (eks. 1000) blir programmet dessuten mer effektivt, hvis **if**-setningen utføres *en* gang, og ikke 1000 ganger.

Kapittel 5-7

Oppgave 1

Funksjonen **lesLovligSvar( )**:

Alle bokstaver (a-z) er like gode som testdata her. Ingen spesielle ytter-grenser, men tester at omformingen mellom store og små bokstaver blir riktig.

Datasett 1: a z som de to bokstavene. Prøver først s. Skal gi meldingen: "Feil! Skriv A, a, Z eller z: ". Deretter skrives A, som er resultatet.

Datasett 2: A Z som de to bokstavene. Prøver først s. Skal gi meldingen: "Feil! Skriv A, a, Z eller z: ". Deretter skrives z, som er resultatet (stor Z).

Funksjonen virker ikke for æ, ø og å, på grunn av at **tolower( )** og **toupper( )** brukes.

Funksjonen **lesTallIIntervall( )**:

Tre typer intervall testes: Ett der nedre grense er mindre enn øvre, ett der de er like og ett der øvre grense er mindre enn nedre (grensene skal da byttes om). Tester også at det kommer en melding dersom tall utenfor intervallet skrives.

Datasett 1: Intervallet [0, 10]. Prøv 0 (nedre grense testes).

Datasett 2: Intervallet [5,5]. Prøv 13, skal gi meldingen: "Ugyldig verdi. Skriv et tall i intervallet [5,5]". Prøv 5.

Datasett 3: Intervallet [10, 0]. Prøv 10 (øvre grense testes).

funksjonene som skal testes legges inn her

```
int main()
{
  int antall;
  cout << "Antall ganger lesLovligSvar() skal testes: ";
  cin >> antall;
  int teller;
  for (teller = 0; teller < antall; teller++) {
    char bokstav1;
    char bokstav2;
    cout <<
     "Hvilke to bokstaver skal funksjonen testes for? ";
    cin >> bokstav1 >> bokstav2;
    char svar = lesLovligSvar(bokstav1, bokstav2);
    cout << "Svaret ble: " << svar << endl;
    }
  cout << "Antall ganger lesTallIIntervall() skal testes: ";
  cin >> antall;
  for (teller = 0; teller < antall; teller++) {
    int nedre;
    int ovre;
    cout << "Oppgi grensene? ";
    cin >> nedre >> ovre;
    int svar = lesTallIIntervall(nedre, ovre);
    cout << "Svaret ble: " << svar << endl;
    }
  return 0;
} // main
```

Oppgave 2

Testdata ("Resultat" er summen av oddetallene i intervallet):

Datasett 1: Intervallet [2,2]. Resultat: 0. Ingen gjennomløp av løkka.
Datasett 2: Intervallet [3,3]. Resultat: 3. Ett gjennomløp av løkka.
Datasett 3: Intervallet [2,6]. Resultat: 8. Nedre grense partall.
Datasett 4: Intervallet [3,5]. Resultat: 8. Nedre grense oddetall.

Vi nøyer oss med å sette opp gjennomgang av datasett 3:

"boks" fra flytskjemaet	sum	teller	Nedre grense	Øvre grense
			2	6
Er **nedre** et partall? *ja*				
Øk **nedre** med 1			3	

teller settes lik **nedre**. **sum** settes lik 0	0	3		
teller <= **ovre**? *ja*				
Øk **sum** med verdien til **teller**	3			
Øk **teller** med 2		5		
teller <= **ovre**? *ja*				
Øk **sum** med verdien til **teller**	8			
Øk **teller** med 2		7		
teller <= **ovre**? *nei*				
Skriv ut **sum**				

Resultatet blir 8, som stemmer med testdatasettet.

Kapittel 6-1

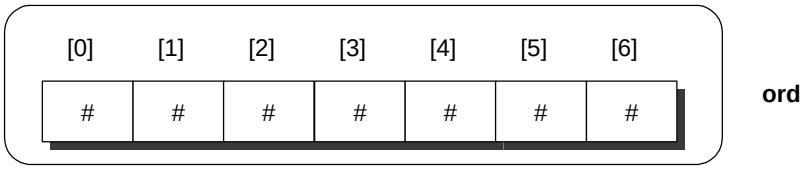
Oppgave 1

```
string ord1;
string ord2;
string ord3;
cout << "Skriv tre ord: ";
cin >> ord1 >> ord2 >> ord3;
```

Oppgave 2

```
string linje = "En to tre";
cout << linje << endl;
```

Oppgave 3



Oppgave 4

Vi kan fylle opp posisjon 0, 1, 2, 3, 4, 5, 6: Bruker ei løkke ettersom vi skal legge inn bokstaver i rekkefølge fra A og utover:

```
string ord(7, '#');
char tegn = 'A';
int pos;
for (pos = 0; pos < ord.length(); pos++) {
  ord[pos] = tegn;
  tegn++;
  }
cout << "Ord: " << ord << endl;
```

Utskriften blir:

```
Ord: ABCDEFG
```

Oppgave 5

```
cout << "\n\n\n\n\n\n";
```

Oppgave 6

Det er ikke mulig å opprette et strengobjekt med plass til tre tegn uten å angi et fylltegn. Deklarasjonen kan f.eks. se slik ut:

```
string tekst(3, ' ');
```

Oppgave 7

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  // a)
  string ord1;
  string ord2;
  cout << "Skriv to ord: ";
  cin >> ord1 >> ord2;

  // b)
  if (ord1 > ord2) { // bytter om rekkefølgen
    string hjelp = ord1;
    ord1 = ord2;
    ord2 = hjelp;
    }
  cout << "Sortert: " << ord1 << " " << ord2 << endl;

  // c)
  string setning = ord1 + " " + ord2 + ".";
  cout << "Setning: " << setning << endl;

  // d)
  cout << "Lengden til ord1 er " << ord1.length() << endl;
  cout << "Lengden til ord2 er " << ord2.length() << endl;
  cout << "Lengden til setning er " << setning.length() <<
          endl;

  // e)
  string setning2 = setning;
  cout << "Setning2: " << setning2 << endl;

  // f)
  if (setning2.length() > 10) {
    setning2[8] = 'x';
    setning2[9] = 'x';
```

```
    setning2[10] = 'x';
    }
  cout << "Setning2: " << setning2 << endl;
  cout << "Setning: " << setning << endl;

  return 0;
} // main
```

Eksempel på kjøring av programmet:

```
Skriv to ord: programmering introduksjon
Sortert: introduksjon programmering
Setning: introduksjon programmering.
Lengden til ord1 er 12
Lengden til ord2 er 13
Lengden til setning er 27
Setning2: introduksjon programmering.
Setning2: introdukxxxn programmering.
Setning: introduksjon programmering.
```

Kapittel 6-2

Oppgave 1

Datatypen til første argument må endres fra **char** til **string**. Endrer også navnet fra **soekeTegn** til **soekeTekst**. De eneste endringene i funksjonskroppen blir dermed ei løkke rundt den eksisterende løkka. Hvert gjennomløpe av denne ytre løkka sjekker ett tegn i **soekeTekst**.

```
int antForekomst(
  string soekeTekst, // Inn
  string ord)        // Inn
{
  int antall = 0;
  int teller1;
  for (teller1 = 0; teller1 < soekeTekst.length(); teller1++) {
    char soekeTegn = soekeTekst[teller1];
    int teller;
    for (teller = 0; teller < ord.length(); teller++) {
      if (ord[teller] == soekeTegn) antall++;
      }
    }
  return antall;
}
```

Kapittel 6-3

Oppgave 1

a) Feil. **find( )** finner kun første forekomst.

b) Riktig.

c) Feil. Eller – det *kan* være riktig, det kommer an på kompilatoren. Vi bør bruke konstanten **string::npos** dersom vi trenger å sjekke på dette.

d) Riktig.

e) Jo, ved å bruke \. Eksempel: "\"".

f) Feil. **replace( )** søker ikke etter teksten den skal bytte ut. Til det må vi bruke **find( )** eller **rfind( )**.

g) Feil. Tredje argument i **replace( )** er en tekst og ikke et enkelt tegn.

h) Generelt er dette utsagnet feil. Men ved å sette antall tegn som skal skiftes ut i **replace( )** til 0, vil **replace( )** fungere som **insert( )**. I programliste 6-6 kan **insert( )**-setningen byttes ut med **tekst.replace(pos, 0, "**");**

i) Riktig.

Oppgave 2

```
Resultat 1: 29
Resultat 2: 6
Resultat 3: 27
Resultat 4: Dettyav flere oppgaver.
Resultat 5: ettyav flere oppgaver.
Resultat 6: ettyav
flere oppgaver.
Resultat 7: ett
Resultat 8: ettyav
flere oppgaver.
```

Oppgave 3

I 1.opplag av boka (2003) er det en trykkfeil i oppgaveteksten. Kan ikke ha pluss mellom tekstkonstanter i en initiering. Den må derfor se slik ut:

```
string tekst =
   "Det er 9.august i dag. Om to uker er skolen i full gang.";
```

a)

```
int pos1 = tekst.find('.', 0);  pos1 blir 8
int pos2 = tekst.find('.', pos1 + 1); pos2 blir 21
int pos3 = tekst.find('9', 0); pos3 blir 7
int pos4 = tekst.find("Om", 0); pos4 blir 23
```

b)

Skriver ut resultatet fra søket:

```
cout << tekst.find("Om", pos4 + 1) << endl;
```

Resultatet er kompilatoravhengig. Microsoft Visual C++ gir 4294967295.

c)

```
int pos = tekst.rfind('.', tekst.length() - 1);
while (pos != string::npos) {
  cout << "punktum funnet på posisjon " << pos << endl;
  pos = tekst.rfind('.', pos - 1);
  }
```

Utskriften ser slik ut:

```
punktum funnet på posisjon 55
punktum funnet på posisjon 21
punktum funnet på posisjon 8
```

Kapittel 6-4

Oppgave 1

Her er funksjonen med et enkelt testprogram:

```
int minStrlen(const char *tekst)
{
  int indeks = 0;
  while (tekst[indeks] != '\0') indeks++;
```

```
  return indeks;
}  // minStrlen

int main()
{
  string ord;
  cout << "Skriv et ord: ";
  cin >> ord;
  const char* ordet = ord.c_str();
  int lengde = minStrlen(ordet);
  cout << "Lengden er " << lengde << endl;
  return 0;
} // main
```

Kapittel 6-5

Oppgave 1

>> leser inn bare ett ord av gangen. Vi vet ikke hvor mange ord hvert navn består av. **getline( )** vil lese inn ei hel linje av gangen.

Oppgave 2

```
int main()
{
  string tekst1;
  string tekst2;
  int tall1;
  int tall2;
  int tall3;

  getline(cin, tekst1);
  cin >> tall1 >> tall2 >> tall3;
  cin.ignore(100, '\n');
  getline(cin, tekst2);

  cout << tekst1 << endl;
  cout << tall1 << " " << tall2 << " " << tall3 << endl;
  cout << tekst2 << endl;
  return 0;
} // main
```

Kjøreeksempel:

```
dette er tekst1
34 23 1
dette er tekst2
dette er tekst1
34 23 1
dette er tekst2
```

Kapittel 6-6

Oppgave 1

```
#include <sstream>

string lagUtskrift(
  string fornavn,     // Inn
  string etternavn,   // Inn
  string postnr,      // Inn
  string poststed,    // Inn
  string fdato)       // Inn
{
  ostringstream utskrift;
  string navn = fornavn + " " + etternavn;
  string postadr = postnr +  " " + poststed;
  utskrift << setw(16) << left << navn << setw(16) <<
              poststed << "Født " << fdato << endl;
  return utskrift.str();
} // lagUtskrift
```

Hovedprogram:

```
int main()
{
  cout << lagUtskrift("Eva", "Nilsen",
                      "7005", "Trondheim", "220380");
  cout << lagUtskrift("Kari Anne", "Sand",
                      "1750", "Halden", "210677");
  cout << lagUtskrift("Nils", "Jonsen",
                      "5002", "Bergen", "130367");
  return 0;
} // main
```

Oppgave 2

```
int main()
{
  string tekst;
  cin >> tekst;

  int sum = 0;

  int ind = 0; // indeksen vi behandler
  while (ind < tekst.length()) {
    // Hopper over x'er
    while (ind < tekst.length() && tekst[ind] == 'x') ind++;
    int tallStart = ind;
    if (ind < tekst.length()) {  // tall funnet
      // Finner sifrene som tallet består av:
      while (ind < tekst.length() &&
             tekst[ind] >= '0' && tekst[ind] <= '9') ind++;
      int tallSlutt = ind;
      string etTall =
        tekst.substr(tallStart, tallSlutt - tallStart + 1);
      int tall = atoi(etTall.c_str());
      cout << "Tall funnet: " << tall << endl;
      sum += tall;
```

```
      }
    }
  cout << "Summen er " << sum << endl;
  return 0;
} // main
```

Kapittel 6-7

Oppgave 1

Initiell algoritme: "Å beregne materialforbruk"

Skriv instruksjoner
Les data
Beregn **forbruk1** (den ene veien)
Beregn **forbruk2** (den andre veien)
Finn **minsteForbruk**
Skriv **minsteForbruk**

Nivå 2:
Detaljering av "Beregn **forbruk1**"

antallBredder = **lengde / golvbeleggBredde**
Hvis noe av golvet er udekket
  Øk **antallBredder** med 1
forbruk1 = antallBredder * lengde

Nivå 2:
Detaljering av "Beregn **forbruk2**"

Tilsvarende "Beregn **forbruk1**", men bytt om **bredde** og **lengde**.

Nivå 2:
Detaljering av "Finn **minsteForbruk**"

Hvis **forbruk1 < forbruk2**
  minsteForbruk = forbruk1
Ellers
  minsteForbruk = forbruk2

Kapittel 7-1

Oppgave 2

```
x er 5 og y 8
x er 6 og y er -13
x er 6 og y er 6
x er -13 og y er 6
```

Kapittel 7-2

Oppgave 1

a)

```
5 10
10 20
20 40
10 20
```

b)

```
5 10
2 4
5 10
2 4
```

Oppgave 2

```
void sorter(
    int &a,   // Inn/ut
    int &b,   // Inn/ut
    int &c)   // Inn/ut
{
    int hjelp;
    if (a > b) {
        hjelp = a;
        a = b;
        b = hejp;
    }
    if (a > c) {
        hjelp = a;
        a = c;
        c = hjelp;
    }
    if (b > c) {
        hjelp = b;
        b = c;
        c = hjelp;
    }
}  // sorter
```

Kommentar: Etter å ha lest kapittel 7-3 vil det være naturlig å bruke funksjonen **byttVerdi()**

Kapittel 7-3

Oppgave 1

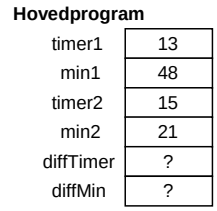
a)

```
const int minPrTime = 60;

//----------------------------------------
int fraTimer(
    int timer,     // Inn
    int minutter)  // Inn
{
    return timer * minPrTime + minutter;
} // fraTimer

//----------------------------------------
void tilTimer(
    int minutter,  // Inn
    int &totTime,  // Ut
    int &totMin)   // Ut
{
    totTime = minutter / minPrTime;
    totMin = minutter % minPrTime;
} // tilTimer
```

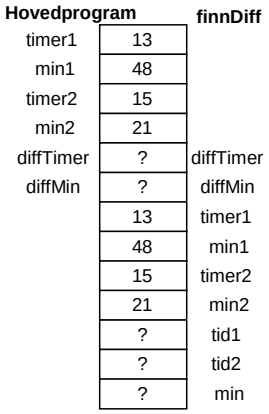
b)

```
//----------------------------------------
void finnDiff(
    int  timer1,   // Inn
    int  min1,     // Inn
    int  timer2,   // Inn
    int  min2,     // Inn
    int &diffTimer,// Ut
    int &diffMin)  // Ut
{
    int tid1 = fraTimer(timer1, min1);
    int tid2 = fraTimer(timer2, min2);
    int diff = abs(tid1 – tid2);
    tilTimer(diff, diffTimer, diffMin);
} // finnDiff
```

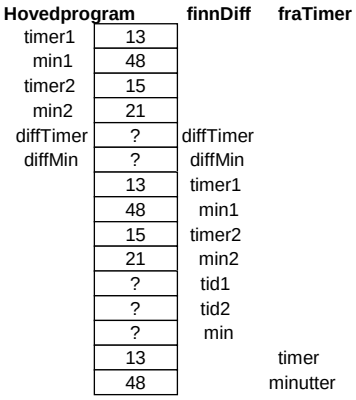
c)



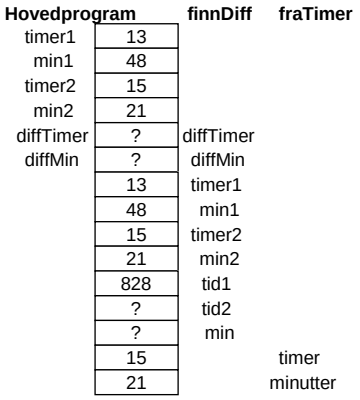
A: I hovedprogrammet



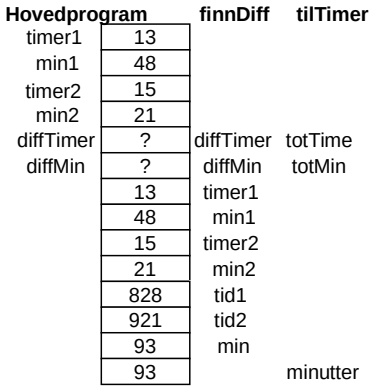
B: I begynnelsen av **finnDiff( )**



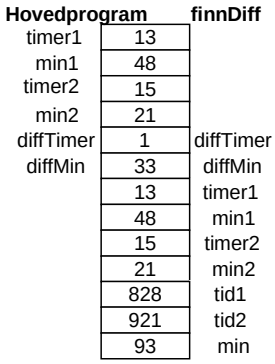
C: Ved begynnelsen av **fraTimer(timer1, min1)**



D: Ved begynnelsen av **fraTimer(timer2, min2)**

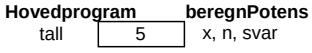


E: Ved begynnelsen av **tilTimer( )**

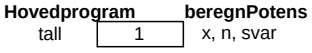


F: Ved slutten av **finnDiff( )**

Oppgave 2



Ved begynnelsen av **beregnPotens( )**



B: Etter setningen **svar = 1;**

Siden både **x**, **n** og **svar** refererer til **tall**, vil vi få utregnet 1 opphøyd i første potens.

Kapittel 7-4

Oppgave 1

For å skille mellom to funksjoner med samme navn, må prototypene være ulike. Kompilatoren ignorerer argumentnavnene i prototypen, derfor må det være forskjell i antall eller type av argumentene.

Oppgave 2

Prototyper:

```
void sorter(int &a, int &b);
void sorter(double &a, double &b);
void sorter(int &a, int &b, int &c);
void sorter(double &a, double &b, double &c);
```

Kapittel 7-5

Oppgave 1

Prototyp:

```
void oek(int &sum, int mengde = 1);
```

Definisjon:

```
void oek(
  int &sum,        // Inn/ut
  int mengde)      // Inn
{
  sum = sum + mengde;
} // oek
```

Oppgave 2

Prototyp:

```
void multiplikasjonstabell(int tilX = 10, int tilY = 10,
                           int fraX = 1, int fraY = 1);
```

Noen mulige funksjonskall:

```
multiplikasjonstabell();             fra 1x1 til 10x10
multiplikasjonstabell(5, 8);         fra 1x1 til 5x8
multiplikasjonstabell(10, 10, 3);    fra 3x1 til 10x10
multiplikasjonstabell(3);            fra 1x1 til 3x10
multiplikasjonstabell(6, 7, 2, 3);   fra 2x3 til 6x7
```

Kapittel 8-1

Oppgave 1

```
a)  ((5 + (8 * tall)) - 3) - (4 / 3)
```

b) `((a < b) && (b > 5)) || ((c == a) && (a != 5))`

c) `–b + ((sqrt((b * b) – (4 * a * c)) / 2) * a)`

Oppgave 2

a) 41

b) `a < b og b > 5 og (a < b) && (b > 5)`

c) Nummererer uttykkene på følgende måte:

```
1   8 * tall
2   5 + (8 * tall)
3   (5 + (8 * tall)) – 3
4   4 / 3
5   ((5 + (8 * tall)) – 3) – (4 / 3)
```

Uttrykkene kan regnes ut i følgende rekkefølger:

```
1, 2, 3, 4, 5
1, 2, 4, 3, 5
1, 4, 2, 3, 5
4, 1, 2, 3, 5
```

d) `(-b + sqrt(b * b – 4 * a * c)) / (2 * a)`

Kapittel 8-2

Oppgave 1

c) (Gir også svar på a og b)

```
//-----------------------------------------------------
//
// Program som kontrollerer svarene i oppgave 8-2-1
//
#include <iostream>
using namespace std;

int main() {
  int tall = 3;
  int a = 0;
  int b = 5;
  int c = -2;
  int d = 1;

  int res = 5 + 8 * tall - 3 - 4 / 3;
```

```
  cout << "Oppgave i: " << res << endl;

  res = tall += 5 > 8;
  cout << "Oppgave ii: " << res << endl;

  cout << "Oppgave iii: ";
  // cout << tall < 9 == tall - 5 << tall = 3;   syntaksfeil
  cout << (tall < 9 == tall - 5) << (tall = 3) << " ";
  cout << (tall < (9 == tall) - 5) << (tall = 3) << " ";
  cout << (tall < (9 == (tall - 5))) << (tall = 3) << " ";
  cout << ((tall < 9 == tall) - 5) << (tall = 3) << endl;

  res = a < b && b > 5 || c == a && a != 5;
  cout << "Oppgave iv: " << res << endl;

  cout << "Oppgave v: " << endl;
  // a + b = c + d;       syntaksfeil
  res = a + (b = c) + d;
  cout << "res = " << res << " b = " << b << endl;
  res = a + (b = c + d);
  cout << "res = " << res << " b = " << b << endl;
  b = 1;

  cout << "Oppgave vi:"  << endl;
  // a /= c + d *= b % 2; syntaksfeil
  res = (a /= c) + (d *= b % 2);
  cout << "res = " << res << " a = " << a << " d = " << d;
  cout << endl;
  res = (a /= c) + (d *= b) % 2;
  cout << "res = " << res << " a = " << a << " d = " << d;
  cout << endl;
  res = ((a /= c) + (d *= b)) % 2;
  cout << "res = " << res << " a = " << a << " d = " << d;
  cout << endl;
  res = a /= (c + (d *= b) % 2);
  cout << "res = " << res << " a = " << a << " d = " << d;
  cout << endl;
  res = a /= (c + (d *= b % 2));
  cout << "res = " << res << " a = " << a << " d = " << d;
  cout << endl;

  return 0;
} // main

/* Kjøring av programmet

Oppgave i: 25
Oppgave ii: 3
Oppgave iii: 03 03 03 -53
Oppgave iv: 0
Oppgave v:
res = -1 b = -2
res = -1 b = -1
Oppgave vi:
res = 1 a = 0 d = 1
res = 1 a = 0 d = 1
res = 1 a = 0 d = 1
res = 0 a = 0 d = 1
res = 0 a = 0 d = 1
*/
```

Oppgave 2

```
//-------------------------------------------------------
//
// Program som finner ut hvor mye plass int og double tar
//
#include <iostream>
using namespace std;

int main() {
  cout << "int bruker " << sizeof(int) << " byte" << endl;
  cout << "double bruker " << sizeof(double) << " byte";
  cout << endl;

  return 0;
} // main

/* Kjøring av programmet

int bruker 4 byte
double bruker 8 byte

*/
```

Kapittel 8-3

Oppgave 1

```
bool erSiffer(
  char tegn,   // Inn
  int &verdi)  // Ut
{
  if (tegn >= '0' && tegn <= '9') {
    verdi = tegn - '0';
    return true;
    }
  else return false;
} // erSiffer
```

Oppgave 2

```
enum sammenlikn {mindre, lik, stoerre};

sammenlikn sjekk(
  int verdi1,  // Inn
  int verdi2)  // Inn
{
  if (verdi1 < verdi2) return mindre;
  else if (verdi1 > verdi2) return stoerre;
  else return lik;
} // sjekk
```

Kapittel 8-4

Oppgave 1

a) 'A' omformes til **int** og multipliseres med 3. Svaret omformes til **double** og adderes til 5.3. 5F omformes fra **float** til **double** og adderes til svaret.Svaret omformes til **long int** og plasseres i **tall**.

b) 3 blir omformet til **true** og 4 % 2 til **false**. 3 && 4 % 2 gir derfor svaret **false**. Vi ber om å omforme dette til **bool**, men siden det allerede er av denne typen, skjer ingen omforming ved "castingen".

c) Siden **tall** er av typen **long int**, omformes både **ok** og 3 til **long int**. Uttrykket på høyre side vil derfor bli **long int**-verdien 0, som omformes til **float**-verdien 0.0 og settes inn i **tallB**.

Kapittel 9-1

Oppgave 1

```
a)  const int antMnd = 12;
    int dagerPrMnd[antMnd];
b)  int mnd;
    for (mnd = 0; mnd < antMnd; mnd++) {
      cin >> dagerPrMnd[mnd];
      }
```

Oppgave 2

b, c og e

Kapittel 9-2

Oppgave 1

```
//-------------------------------------------------------
//
// Program som beregner antall dager mellom to datoer
//
#include <iostream>
using namespace std;

const int antMnd = 12;
int main() {
```

```
  int dagerPrMnd[antMnd] = {31, 28, 31, 30, 31, 30 , 31,
                            31, 30, 31, 30, 31};

  int mnd1;
  int dag1;
  int mnd2;
  int dag2;

  cout << "Angi måned og dagnr for første dato: ";
  cin >> mnd1 >> dag1;
  cout << "Angi måned og dagnr for andre dato: ";
  cin >> mnd2 >> dag2;

  int antDager = -dag1;
  for (mnd = mnd1; mnd <mnd2; mnd++) {
    antDager += dagerPrMnd[mnd];
     }
  antDager += dag2;

  cout << "Antall dager: " << antDager << endl;

  return 0;
} // main
```

Oppgave 2

```
a)  const int antVarer = 200;
    double innPris[antVarer];
    double utPris[antVarer];
    int antEnheter[antVarer];

b)  int nr;
    cout << "Oppgi innkjøpspris og antall på lager" << endl;
    for (nr = 0; nr < antvarer; nr++) {
      cout << "Varenr " << nr << ": ";
      cin >> innPris[nr] >> antallEnheter[nr];
      }

c)  const int avanse = 0.5;
    for (nr = 0; nr < antVarer; nr++) {
      utPris[nr] = (1 + avanse) * innPris[nr];
      }

d)  const int       antRed = 4;
    const double reduksjon = 0.25;
    cout << "Angi varenr for tilbudsvarene: ";
    int redNr = 0;
    while (redNr < antRed) {
      cin >> nr;
      if (nr >= 0 && nr < antVarer) {
        utPris[nr] -= utPris[nr] * reduksjon;
        redNr++;
        }
      else cout << "Ulovlig varenummer!" << endl;
      } // while
```

```
e)  double fortjeneste = 0.0;
    for (nr =0; nr < antVarer; nr++) {
      fortjeneste += utPris[nr] – innPris[nr];
      }
```

Oppgave 3

```
//---------------------------------------------------------
//
// Program som skriver ord i motsatt rekkefølge
//
#include <iostream>
#include <string>
using namespace std;

const int maksAntOrd = 100;
int main() {
  string ordene[maksAntOrd];

  cout << "Skriv noen ord, avslutt med ordet #" << endl;
  string ord;
  cin >> ord;
  int nr = 0;
  while (nr < maksAntOrd && ord != "#") {
    ordene[nr] = ord;
    cin >> ord;
    nr++;
  }

  cout << "Her er ordene i motsatt rekkefølge: " << endl;
  int antOrd = nr;
  for (nr = antOrd -1; nr >= 0; nr--) {
    cout << ordene[nr] << " ";
  }
  cout << endl;

  return 0;
} // main
```

Oppgave 4

```
//---------------------------------------------------------
//
// Program som lager telefonliste
//
#include <iostream>
#include <string>
using namespace std;

const int maksNavn = 100;
const int maksNavnelengde = 30;
int main() {
  string navnene[maksNavn];
  string tlfnr[maksNavn];
```

```
  cout << "Skriv navn og telefonnummer, ett på hver linje, ";
  cout << "avslutt med ordet #";
  cout << endl;
  string linje;
  getline(cin, linje);
  int nr = 0;
  while (nr < maksNavn && linje[0] != '#') {
    int pos = linje.rfind(' ', linje.length() - 1);
    navnene[nr] = linje.substr(0, pos);
    tlfnr[nr] = linje.substr(pos + 1);
   getline(cin, linje);
    nr++;
    }

  cout << "Her er liste over navn og telefonnummer: " << endl;
  int antNavn = nr;
  for (nr = 0; nr < antNavn; nr++) {
    string blank(maksNavnelengde - navnene[nr].length(), ' ');
    cout << navnene[nr] << blank << tlfnr[nr] << endl;
    }
  cout << endl;

  return 0;
} // main
```

Oppgave 5

```
const int antMnd = 12;
int arbDager[antMnd] =
  {22, 20, 21, 19, 20, 20, 23, 21, 22, 23, 20, 21};
string navn =
  {"januar", "februar", "mars", "april", "mai", "juni", "juli",
    "august", "september", "oktober", "november", "desember"};
int nr;
for (nr = 0; nr < antMnd; nr++) {
  cout << "I " << navn[i] << " er det " << arbDager[i];
  cout << " arbeidsdager" << endl;
  }
```

Kapittel 9-3

Oppgave 1

Maksimal lengde: a: 31, b: 600, c: 366

Aktuell lengde: a: 28 – 31, b: 400 – 600, c: 300 - 366

Oppgave 2

```
//-------------------------------------------------------
//
// Program som finner miderste element(er)
//
#include <iostream>
#include <string>
using namespace std;

const int maks = 100;
int main() {
  int tallene[maks];

  int aktuellLengde;
  cout << "Hvor mange tall skal behandles? ";
  cin >> aktuellLengde;

  cout << "Skriv tallene: " << endl;
  int teller;
  for (teller = 0; teller < aktuellLengde; teller++) {
    cin >> tallene[teller];
    }

  int midt = aktuellLengde / 2;
  if (aktuellLengde % 2 != 0) {
    cout << "Det midterste tallet er " << tallene[midt];
    }
  else {
    cout << "De to midterste tallene er ";
    cout << tallene[midt - 1] << " og " << tallene[midt];
    }
  cout << endl;

  return 0;
} // main
```

Kapittel 9-4

Oppgave 1

```
//-------------------------------------------------------
//
// Program som finner totalpris
//
#include <iostream>
using namespace std;

const int maksAntall = 50;

void multTabell(int aktLengde, const int *tab1,
                const int *tab2, int *res);
void lesTabell(int maksLengde, int &aktLengde, int *res);
```

```
void skrivTabell(int aktLengde, int *tab);

int main() {
  int priser[maksAntall];
  int antallEnheter[maksAntall];
  int totalPris[maksAntall];
  int antVarer;
  int antPriser;

  lesTabell(maksAntall, antVarer, antallEnheter);
  lesTabell(maksAntall, antPriser, priser);

  if (antVarer == antPriser) {
     multTabell(antVarer, antallEnheter, priser, totalPris);
     skrivTabell(antVarer, totalPris);
    }
  else cout << "Ikke like lange tabeller" << endl;

  return 0;
} // main
//-----------------------------------------------------
//
// Funksjon som multipliserer samhørende elemetner i to
// tabeller
//
void multTabell(
  int  aktLengde,     // Inn
  const int *tab1,    // Inn
  const int *tab2,    // Inn
  int       *res)     // Ut
{
  int teller;
  for (teller = 0; teller < aktLengde; teller++) {
    res[teller] = tab1[teller] * tab2[teller];
    }
} // multTabell
```

lesTabell() og **skrivTabell()** som i kapittel 9.4

Oppgave 2

```
//-----------------------------------------
//
// Program for mosjonsløp
//
#include <iostream>
#include <cmath>
using namespace std;

const int maksAntall = 50;

int finnNaermeste(double verdi, int aktLengde,
                  const double *tabell);
void lesTider(int maksLengde, int &aktLengde, double *tabell);
double beregnSnitt(int aktLengde, const double *tab);
```

```
int main() {
  double tider[maksAntall];
  int antDeltakere;

  cout << "Angi tider - negativt for å avslutte" << endl;
  lesTider(maksAntall, antDeltakere, tider);
  double snitt = beregnSnitt(antDeltakere, tider);
  cout << "Idealtid: " << snitt << endl;
  int vinner = finnNaermeste(snitt, antDeltakere, tider);
  cout << "Vinner ble startnummer " << vinner + 1 << endl;

  return 0;
} // main
//-----------------------------------------------------
//
// Funksjon som leser inn resultattider
//
void lesTider(
  int     maksLengde,     // Inn
  int    &aktLengde,      // Ut
  double *tabell)         // Ut
{
  double tid;
  aktLengde = 0;

  cin >> tid;
  while (aktLengde < maksLengde && tid > 0.0) {
    tabell[aktLengde] = tid;
   aktLengde++;
   cin >> tid;
  }
} // lesTider
//-----------------------------------------------------
//
// Funksjon som beregner gjennomsnittet
//
double beregnSnitt(
  int          aktLengde, // Inn
  const double *tabell)   // Inn
{
  if (aktLengde > 0) {
   double sum = 0.0;
   int teller;
   for (teller = 0; teller < aktLengde; teller++) {
     sum += tabell[teller];
     }
   return sum / aktLengde;
  }
  else return 0.0;
} // beregnSnitt
//-----------------------------------------------------
//
// Funksjon som finner den som kommer nærmest en bestemt tid
//
int finnNaermeste(
  double        verdi,    // Inn
  int           aktLengde,    // Inn
  const double *tab)      // Inn
{
```

```
  int hittilNaermest = 0;
  double diff = fabs(tab[0] - verdi);
  int teller;
  for (teller = 1; teller < aktLengde; teller++) {
    if (fabs(tab[teller] - verdi) < diff) {
      hittilNaermest = teller;
      diff = fabs(tab[teller] - verdi);
      }
    }
  return hittilNaermest;
} // finnNaermest
```

Oppgave 3

Ved å forandre > til >= når vi sammenlikner tabellelementene.

Oppgave 4

```
//-------------------------------------------------------
//
// Funksjon som finner alle de største verdiene
//
void finnStoerst(
  int        aktLengde,     // Inn
  const int *tabell,        // Inn
  int        maksIndekser,  // Inn
  int       *resultat,      // Ut
  int       &antall)        // Ut
{
  resultat[0] = 0;
  antall = 1;
  int verdi = tabell[0];
  int teller = 0;
  for (teller = 1; teller < aktLengde; teller++) {
    if (tabell[teller] > verdi) {
      antall = 1;
      verdi = tabell[teller];
      resultat[0] = teller;
      }
    else if (tabell[teller] == verdi) {
      resultat[antall] = teller;
      antall++;
      }
    }
} // finnStoerst
```

Kapittel 9-5

Oppgave 1

```
//-------------------------------------------------------
//
// Funksjon som søker etter et ord i en usortert tabell
//
bool sekvSoekOrd(
  int           antall,       // Inn
  const string *tabell,       // Inn
  string        letEtter,   // Inn
  int          &indeks)       // Ut
{
  int teller = 0;
  while (teller < antall
      && tabell[teller] != letEtter) teller++;
  if (teller < antall) {
    indeks = teller;
    return true;
    }
  else return false;
} // sekvSoekOrd
```

Oppgave 2

0 gjennomløp av løkka, og funksjonen returnerer **false**.

løkka går til **teller** blir lik **antall**, og funksjonen returner **false**.

Oppgave 3

I stedet for **else return false;** skriver vi

```
else {
  indeks = teller;
  return false;
  }
```

Oppgave 4

```
// indeks er indeksen til det som skal fjernes

// sortert tabell
int teller = indeks + 1;
while (teller < antall) {
```

```
  tabell[teller – 1] = tabell[teller];
  teller++;
  }
antall--;

// usortert tabell
tabell[indeks] = tabell[antall – 1 ];
antall--;
```

Kapittel 9-6

Oppgave 1

a)

```
const int maksAntLag = 14;
const int muligeRes = 3;
int resultater[maksAntLag][muligeRes];
```

b)

```
const int poengPrKamp[] = {3, 1, 0};
//-------------------------------------------------------
//
// Funksjon som finner antall poeng i en fotballserie
//
void beregnPoeng(
  const int resultat[maksAntLag][muligeRes],    // Inn
  int        antall,      // Inn
  const int *poengPrKamp, // Inn
  int        muligeRes,   // Inn
  int       *poenger)     // Ut
{
  int lagNr;
  int resNr;
  for (lagNr = 0; lagNr < antall; lagNr++) {
    poenger[lagNr] = 0;
    for (resNr = 0; resNr < muligeRes; resNr++) {
      poenger[lagNr] +=
        resultat[lagNr][resNr] * poengPrKamp[resNr];
      }
    }
} // beregnPoeng
```

Oppgave 2

a)

```
  const int antallFelt = 3;
  const int antallTre  = 10;
```

```
const int antallAar  = 5;
int epler[antallFelt][antallTre][antallAar];
```

b)

```
int total[antallFelt];
int felt;
for (felt = 0; felt < antallFelt; felt++) {
  total[felt] = 0;
  int tre;
  for (tre = 0; tre < antallTre; tre++) {
    int aar;
    for (aar = 0; aar < antallAar; aar++) {
      total[felt] += epler[felt][tre][aar];
      }
    }
  }
```

Kapittel 10-1

Oppgave 1

Merknad: Et kontonummer er vanligvis på 11 siffer. I vår kompilator er største tillatte verdi for long int 2 147 483 647. Tallområdet er derfor ikke stort nok for et virkelig kontonummer. (En kan bruke et strengobjekt.)

Utvidelse i klassedefinisjonen:

```
class Konto {
  public:
    void     settKontonr(
               long int nyttKontonr);  // Inn
    long int finnKontonr() const;
```

Utvidelse i implementasjonen:

```
//-------------------------------------------------------------
void Konto::settKontonr(long int nyttKontonr)
{
  kontonr = nyttKontonr;
}  // settKontonr
//-------------------------------------------------------------
long int Konto::finnKontonr() const
{
  return kontonr;
} // finnKontonr
```

Utvidelse i testprogrammet:

```
konto1.settKontonr(523456789L);
```

```
cout << "Kontonummeret er " << konto1.finnKontonr() << endl;
```

Oppgave 2

Må ha følgende include-kommandoer i filen konto.cpp:

```
#include <string>
#include <sstream>
#include <iomanip>
using namespace std;
```

Klassedefinisjonen utvides slik:

```
string lagUtskrift() const;
```

Her er implementasjonen:

```
string Konto::lagUtskrift() const
{
  ostringstream utskrift;
  utskrift.setf(ios::fixed, ios::floatfield);
  utskrift << setprecision(2) << showpoint;
  utskrift << setw(18) << left << "Kredittgrense kr. " <<
              setw(8) << right << grense << endl <<
              setw(18) << left << "Saldo kr. " <<
              setw(8) << right << saldo << endl;
  return utskrift.str();
} // lagUtskrift
```

I testprogrammet:

```
cout << konto1.lagUtskrift() << endl;
```

Oppgave 3

Objekt: Trekanten.
Operasjoner: Sett og finn grunnlinje og høyde. Finn areal.
Attributter: Grunnlinje og høyde.

Oppgave 4

```
class Trekant {
  public:
    void   settData(
             double nyGrunnlinje,    // Inn
             double nyHoyde);        // Inn
    double finnGrunnlinje() const;
    double finnHoyde() const;
    double finnAreal() const;
```

```
  private:
    double grunnlinje;
    double hoyde;
  }; // Trekant

//-----------------------------------------------------
void Trekant::settData(double nyGrunnlinje, double nyHoyde)
{
  grunnlinje = nyGrunnlinje;
  hoyde = nyHoyde;
} // settData

//-----------------------------------------------------
double Trekant::finnGrunnlinje() const
{
  return grunnlinje;
} // finnGrunnlinje

//-----------------------------------------------------
double Trekant::finnHoyde() const
{
  return hoyde;
} // finnHoyde

//-----------------------------------------------------
double Trekant::finnAreal() const
{
  return grunnlinje * hoyde * 0.5;
} // finnAreal

//-----------------------------------------------------
#include <iostream>
using namespace std;
int main()
{
  Trekant t;
  t.settData(5, 6);
  double g = t.finnGrunnlinje();
  double h = t.finnHoyde();
  cout << "Grunnlinje " << g << ", og høyde " << h
    << " gir areal: " << t.finnAreal() << endl;
  return 0;
} // main
```

Kapittel 10-2

Oppgave 1

Talloppgaven:

Objekt: Tallet.
Operasjoner: Finn ut om tallet er > 0. Finn ut om tallet er delelig med 2

og 4.
Attributter: Verdien til tallet.

Likningskontoret:

Objekt: Likningskontoret, de forskjellige kontorene der, selvangivelsen, du som person.
Operasjoner: Registrer fødselsdato. Gi kontorene nummer og intervall for dagene. Finn romnr utfra fødselsdato.
Attributter: Fødselsdato, romnr og intervall.

Oppgave 2

To kategorier brukere:

1. Programmerere som bruker klassen som en byggekloss i det programmet de skal lage.

2. Programmene som lages, blir også brukere av klassen.

Det er viktig å gjøre datadelen utilgjengelig for brukerne, slik at implementasjonen av klassen kan forandres uten at brukerne blir berørt.

Kapittel 10-3

Oppgave 1

a) Datatypen **Maanedslengde** er definert som en oppramsingstype inni klassen **Maaned**. Variabler av denne typen kan enten ha verdien **minAntDager** eller **maksAntDager**.

b) Vi må be brukeren skrive inn tallverdien (28 eller 29). Dersom 28 er skrevet inn, kan vi "caste" til datatypen **Maanedslengde**:

```
februar = (Maanedslengde) innlestVerdi;
```

Verdien 29 er imidlertid ikke en gyldig verdi for denne datatypen.

Oppgave 2

a) og b)

```
class Punkt {
  public:                  kolon manglet
    void settX(
           int nyX);//Inn  argument og datamedlem hadde samme navn
    void settY(
           int nyY);//Inn  samme her
    int finnX();           ingenting kom ut av funksjonen (eller inn)
    int finnY();           samme her
  private                  kolon manglet
    int x;
    int y;
  }; // Punkt              semikolon manglet

//-----------------------------------------------------
void Punkt::settX(int nyX)
{
  x = nyX;
} // settX

//-----------------------------------------------------
void Punkt::settY(int nyY)
{
  y = nyY;
} // settY

//-----------------------------------------------------
int Punkt::finnX()
{
  return x;
} // finnX

//-----------------------------------------------------
int Punkt::finnY()
{
  return y;
} // finnY
```

c)

```
#include <iostream>
#include "punkt.cpp"
using namespace std;
int main()
{
  Punkt p;
  p.settX(3);
  p.settY(4);
  cout << "x er " << p.finnX() << " og y er "
```

```
        << p.finnY() << endl;
  return 0;
} // main
```

d)

```
class Punkt {
  public:
    bool settX( //false dersom utenfor [minX, maksX]
              int nyX);      // Inn
    bool settY( //false dersom utenfor [minY, maksY]
              int nyY);      // Inn
    int     finnX();
    int     finnY();

    enum grenser
      {minX = 0, minY = 0, maksX = 25, maksY = 35};

  private:
    int x;
    int y;
  }; // Punkt

//-----------------------------------------------------
bool Punkt::settX(int nyX)
{
  if (nyX >= minX && nyX <= maksX) {
    x = nyX;
    return true;
    }
  else return false;
} // settX


//-----------------------------------------------------
bool Punkt::settY(int nyY)
{
  if (nyY >= minY && nyY <= maksY) {
    y = nyY;
    return true;
    }
  else return false;
} // settY

//-----------------------------------------------------
int Punkt::finnX()
{
  return x;
} // finnX

//-----------------------------------------------------
int Punkt::finnY()
{
  return y;
} // finnY
```

Testprogram:

```
#include <iostream>
#include "punkt2.cpp"
using namespace std;
int main()
{
  const int antallTester = 3;
  int teller;
  for (teller = 0; teller < antallTester; teller++) {
    Punkt p;
    int x;
    int y;
    int x1 = Punkt::minX;
    int x2 = Punkt::maksX;
    int y1 = Punkt::minY;
    int y2 = Punkt::maksY;
    cout << "Skriv x [" << x1 << "," << x2
      << "] og y [" << y1 << "," << y2 << "]: ";
    cin >> x >> y;
    bool okX = p.settX(x);
    bool okY = p.settY(y);
    if (!okX) cout << "x utenfor intervallet\n";
    if (!okY) cout << "y utenfor intervallet\n";
    if (okX) cout << "x var " << p.finnX() << endl;
    if (okY) cout << "y var " << p.finnY() << endl;
    }
  return 0;
} //main

/* Eksempel på kjøring:
Skriv x [0,25] og y [0,35]: 10 15
x var 10
y var 15
Skriv x [0,25] og y [0,35]: 30 6
x utenfor intervallet
y var 6
Skriv x [0,25] og y [0,35]: 40 -4
x utenfor intervallet
y utenfor intervallet
*/
```

Oppgave 3

Klassedefinisjonen:

```
#include <iostream>
using namespace std;

class Konto {

  public:
    ....
    void    nullstill();
    void    les();
    void    skriv() const;
```

Implementasjonen:

```
//-----------------------------------------------------
void Konto::nullstill()
{
  saldo = 0.0;
  grense = 0.0;
} // nullstill

//-----------------------------------------------------
void Konto::les()
{
  cout << "Oppgi kredittgrense: ";
  cin >> grense;
  cout << "Oppgi saldo: ";
  cin >> saldo;
} // les

//-----------------------------------------------------
void Konto::skriv() const
{
  cout << "Kredittgrense: " << grense << endl;
  cout << "Saldo        : " << saldo << endl;
} // skriv
```

Testprogram:

```
int main()
{
  Konto konto1;
  konto1.nullstill();
  konto1.les();
  konto1.skriv();
  return 0;
} // main
```

Kapittel 10-4

Oppgave 1

Klassedefinisjonen:

```
#include <string>
#include "navn.cpp"
using namespace std;

class Konto {
  public:
    void    settNavn(
```

```
              string nyttHeleNavnet); // Inn
    void    settFornavn(
              string nyttFornavn);    // Inn
    void    settMellomnavn(
              string nyttMellomnavn); // Inn
    void    settEtternavn(
              string nyttEtternavn);  // Inn

    string  finnHeleNavnet() const;
    string  finnFornavn() const;
    string  finnMellomnavn() const;
    string  finnEtternavn() const;
    ....
  private:
    Navn navn;
    double grense;
    double saldo;
};
```

Implementasjonen:

```
void Konto::settNavn(string nyttHeleNavnet)
{
  navn.settNavn(nyttHeleNavnet);
} // settNavn

//-------------------------------------------------------------
void Konto::settFornavn(string nyttFornavn)
{
  navn.settFornavn(nyttFornavn);
} // settFornavn

//-------------------------------------------------------------
void Konto::settMellomnavn(string nyttMellomnavn)
{
  navn.settMellomnavn(nyttMellomnavn);
} // settMellomnavn

//-------------------------------------------------------------
void Konto::settEtternavn(string nyttEtternavn)
{
  navn.settEtternavn(nyttEtternavn);
} // settEtternavn

//-------------------------------------------------------------
string Konto::finnHeleNavnet() const
{
  return navn.finnNavn();
}  // finnHeleNavnet

//-------------------------------------------------------------
string Konto::finnFornavn() const
{
  return navn.finnFornavn();
} // finnFornavn

//-------------------------------------------------------------
string Konto::finnMellomnavn() const
{
```

```
  return navn.finnMellomnavn();
} // finnMellomnavn

//-------------------------------------------------------------
string Konto::finnEtternavn() const
{
  return navn.finnEtternavn();
} // finnEtternavn
```

Testprogram med kjøreresultat:

```
#include <iostream>
#include <string>
#include "konto.cpp"
using namespace std;
int main()
{
  Konto konto1;
  konto1.settNavn("Anne Grethe Hansen");
  cout << "Hele navnet: " << konto1.finnHeleNavnet() << endl;
  cout << "Fornavn: " << konto1.finnFornavn() << endl;
  cout << "Mellomnavn: " << konto1.finnMellomnavn() << endl;
  cout << "Etternavn: " << konto1.finnEtternavn() << endl;
  konto1.settFornavn("Beate");
  konto1.settMellomnavn("Marie Kristine");
  konto1.settEtternavn("Stavne");
  cout << "Hele navnet, etter endring: " <<
          konto1.finnHeleNavnet() << endl;
  return 0;
} // main
/* Kjøring av programmet:
Hele navnet: Anne Grethe Hansen
Fornavn: Anne
Mellomnavn: Grethe
Etternavn: Hansen
Hele navnet, etter endring: Beate Marie Kristine Stavne
*/
```

Oppgave 2

Vi har nå kun ett datamedlem:

```
string navn;
```

Da ser implementasjonen slik ut:

```
//-------------------------------------------------------------
void Navn::settNavn(string nyttNavn)
{
  navn = nyttNavn;
} // settNavn
```

```
//-------------------------------------------------------------
void Navn::settFornavn(string nyttFornavn)
{
  navn = nyttFornavn + " " + finnMellomnavn() + " " +
         finnEtternavn();
} // settFornavn

//-------------------------------------------------------------
void Navn::settMellomnavn(string nyttMellomnavn)
{
  navn = finnFornavn() + " " + nyttMellomnavn + " " +
         finnEtternavn();
} // settMellomnavn

//-------------------------------------------------------------
void Navn::settEtternavn(string nyttEtternavn)
{
  navn = finnFornavn() + " " + finnMellomnavn() + " " +
         nyttEtternavn;
} // settEtternavn

//-------------------------------------------------------------
string Navn::finnFornavn() const
{
  // Fornavnet er definert som første ord i navnet
  int forsteBlank = navn.find(' ');
  string fornavn = navn.substr(0, forsteBlank);
  return fornavn;
} // finnFornavn

//-------------------------------------------------------------
string Navn::finnEtternavn() const
{
  // Etternavnet er definert som siste ord i navnet
  int sisteBlank = navn.rfind(' ');
  int lengdeEtternavn = navn.length() - sisteBlank - 1;
  string etternavn =
           navn.substr(sisteBlank + 1, lengdeEtternavn);
  return etternavn;
} // finnEtternavn

//-------------------------------------------------------------
string Navn::finnMellomnavn() const
{
  // Først fornavnet
  int forsteBlank = navn.find(' ');
  string fornavn = navn.substr(0, forsteBlank);

  // Så mellomnavnet
  string mellomnavn;
  int sisteBlank = navn.rfind(' ');
  if (forsteBlank == sisteBlank) mellomnavn = "";
  else {
    int lengdeMellomnavn = sisteBlank - forsteBlank - 1;
    mellomnavn =
        navn.substr(forsteBlank + 1, lengdeMellomnavn);
    }

  return mellomnavn;
} // finnMellomnavn
```

```
//-------------------------------------------------------------
string Navn::finnNavn() const
{
  return navn;
} // finnNavn
```

Kapittel 10-5

Oppgave 1 og 2

Lar funksjonen **sammenlikn( )** (oppgave 1) bruke funksjonen **avstand( )** (oppgave 2).

Klassedefinisjonen:

```
using namespace std;
class Punkt {
  public:
    .. som før ...
    int    sammenlikn(// -1 hvis dette er nærmest origo,
                      // 0 hvis like, +1 hvis det andre
              Punkt detAndre) const; // Inn
    double avstand(
              Punkt detAndre) const; // Inn

  private:
    int x;
    int y;
  }; // Punkt
```

Implementasjonen:

```
#include <cmath>  // fabs()

//-----------------------------------------------------
int Punkt::sammenlikn(Punkt detAndre) const
{
  const double eps = 0.000001;
  Punkt origo;
  origo.settX(0.0);
  origo.settY(0.0);
  double fraOrigoDette = avstand(origo);
  double fraOrigoDetAndre = detAndre.avstand(origo);
  if (fabs(fraOrigoDette - fraOrigoDetAndre) < eps) return 0;
  else if (fraOrigoDette < fraOrigoDetAndre) return -1;
  else return 1;
} // sammenlikn

//-----------------------------------------------------
double Punkt::avstand(Punkt detAndre) const
```

```
{
  int avstandX = x - detAndre.x;
  int avstandY = y - detAndre.y;
  return
    sqrt(avstandX * avstandX + avstandY * avstandY);
} // avstand
```

Oppgave 3

```
int Navn::sammenlikn(Navn detAndre) const
{
  string detteNavnet = finnEtternavn() + " " + finnFornavn() +
                       " " + finnMellomnavn();
  string detAndreNavnet = detAndre.finnEtternavn() + " " +
      detAndre.finnFornavn() + " " + detAndre.finnMellomnavn();
  if (detteNavnet < detAndreNavnet) return -1;
  else if (detteNavnet > detAndreNavnet) return 1;
  else return 0;
} // sammenlikn
```

Medlemsfunksjonene bruker tilgangsfunksjonene for å få tilgang til datadelen av klassen. Det vil si at bare tilgangsfunksjonene blir avhengig av dataoppbygningen. Det blir dermed enda enklere å forandre på dataoppbygningen, enda færre funksjoner å forandre på.

Kapittel 10-7

Oppgave 1

Se løsningen til oppgave 1 og 2, kapittel 10-4.

Oppgave 2

```
//-------------------------------------------------------------
//
// konto_t.cpp
//
// Oppgave 10-7-2
//
#include <iostream>
#include <string>
#include <cmath>
#include "konto.cpp"

using namespace std;

// Pga at vi skal teste at mer enn en konto har maks-verdi,
// setter vi maks antall kontoer lik 5, og ikke 3 slik det står
// i oppgaveteksten
const int maksAnt = 5;  // brukes i alle funksjonene

bool   lesKontoer(int &ant, Konto *kontoer);
```

```
double finnHoyesteSaldo(const Konto *kontoer,int ant);
void   finnIndekserGittSaldo(double saldo, Konto  *kontoer,
         int antKontoer, int *indekser, int &antIndekser);
void   finnIndekserGittSaldo(double saldo, int *indekser,
                             int &ant);
void   skrivNavnGittIndekser(const Konto *kontoer,
        const int *indekser, int antIndekser);

int main()
{
  Konto kontoer[maksAnt];
  int ant;
  if (!lesKontoer(ant, kontoer)) {
    cout << "Ikke plass til flere kontoer\n\n";
  }
  cout << "Antall kontoer registrert: " << ant << endl << endl;

  double maks = finnHoyesteSaldo(kontoer, ant);
  if (maks >= 0.0) { // finner alle med saldo lik høyeste saldo
    int maksInd[maksAnt];
    int antHoyest;
    finnIndekserGittSaldo(maks, kontoer, ant,
                          maksInd, antHoyest);
    cout << "Størst saldo, kr " << maks << " registrert på " <<
            antHoyest << " navn:\n";
    skrivNavnGittIndekser(kontoer, maksInd, antHoyest);
    }
  else {
    cout <<
     "Ingen kontoer registrert, kan ikke finne maks. saldo.\n";
    }
  return 0;
} // main

//-------------------------------------------------------------
//
// Funksjonen returnerer false dersom flere kontoer enn tillatt
// er forsøkt lest inn.
//
bool lesKontoer(
       int   &ant,   // Ut, antall konto-objekter fylt med data
       Konto *kontoer) // Ut, maks. str. global konstant aksAnt
{
  ant = 0;
  string navn;
  cout << "Kontonavn (avslutt med linjeskift): ";
  getline(cin, navn);
  while (ant < maksAnt && navn.length() > 0) {
    double saldo;
    cout << "Saldo: ";
    cin >> saldo;
    kontoer[ant].settNavn(navn);
    kontoer[ant].settSaldo(saldo);
    ant++;
    cin.ignore(100, '\n');
    cout << "Kontonavn (avslutt med linjeskift): ";
    getline(cin, navn);
```

```
    }
  cout << endl;
  if (navn.length() == 0) return true;
  else return false;
}  // lesKontoer

//-------------------------------------------------------------
//
// Finner høyeste saldo.
// Returnerer negativt tall, dersom ingen kontoer registrert.
//
double finnHoyesteSaldo(
         const Konto *kontoer,    // Inn
         int          ant)        // Inn
{
  if (ant > 0) {
    double maks = kontoer[0].finnSaldo();
    for (int i = 1; i < ant; i++) {
      if (kontoer[i].finnSaldo() > maks) {
        maks = kontoer[i].finnSaldo();
        }
      }
    return maks;
    }
  else return -1.0;
} // finnHoyesteSaldo

//-------------------------------------------------------------
//
// Finner indeksene til alle kontoene med en gitt saldo
//
void finnIndekserGittSaldo(
       double  saldo,      // Inn
       Konto  *kontoer,    // Inn
       int     antKontoer, // Inn
       int    *indekser,   // Ut, maks. str. lik maksAnt
       int    &antIndekser)// Ut
{
  antIndekser = 0;
  for (int j = 0; j < antKontoer; j++) {
    if (fabs(kontoer[j].finnSaldo() - saldo) < 0.0001) {
      indekser[antIndekser] = j;
      antIndekser++;
    }
  }
} // finnIndekserGittSaldo

//-------------------------------------------------------------
//
// Skriver ut liste over navn på gitte indekser
// Ingen kontroll av gyldige indekser
//
void skrivNavnGittIndekser(
       const Konto *kontoer,     // Inn
       const int   *indekser,    // Inn
       int          antIndekser) // Inn
{
  for (int k = 0; k < antIndekser; k++) {
    int kIndeks = indekser[k];
    cout << kontoer[kIndeks].finnHeleNavnet() << endl;
```

```
  }
} // skrivNavnGittIndekser


/* Eksempel på kjøring:
Kontonavn (avslutt med linjeskift): Ole Hansen
Saldo: 2500
Kontonavn (avslutt med linjeskift): Per Johnsen
Saldo: 2200
Kontonavn (avslutt med linjeskift): Ingrid Haug
Saldo: 2500
Kontonavn (avslutt med linjeskift): Pernille Aas
Saldo: 2300
Kontonavn (avslutt med linjeskift): Eva Hansen
Saldo: 2500
Kontonavn (avslutt med linjeskift): Turid Favel

Ikke plass til flere kontoer

Antall kontoer registrert: 5

Størst saldo, kr 2500 registrert på 3 navn:
Ole  Hansen
Ingrid  Haug
Eva  Hansen
*/
```

Kapittel 11-1

Oppgave 1

```
ifstream inn;
inn.open("inn.dat");

ofstream ut;
ut.open("ut.dat");
```

Oppgave 2

Dersom filen **"inn.dat"** ikke eksisterer, blir det registrert som feil. Ingen flere operasjoner blir utført på objektet **inn**. Det skrives ikke ut feilmelding av noe slag. Dersom filen **"ut.dat"** ikke eksisterer, blir den laget.

Oppgave 3

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
```

```
int main()
{
  ofstream resultat;
  resultat.open("ut.dat");
  if (!resultat) {
    cout << "Filåpningen gikk ikke bra.\n";
    exit(EXIT_FAILURE);
    }
  ....
```

Oppgave 4

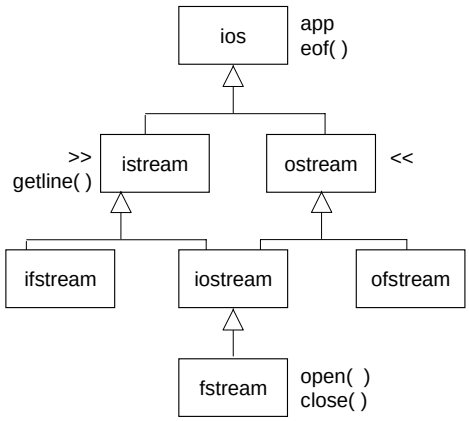
```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
  ofstream resultat;
  resultat.open("ut.dat");
  int tall1 = 1;
  int tall2 = 2;
  int tall3 = 3;
  string navn = "Oline Olsen";
  resultat << tall1 << " " << tall2 << " " << tall3 <<
              " " << navn << " " << endl;
  ...
```

Kapittel 11-3

Oppgave 1

Datamedlemmer og funksjonsmedlemmer arves nedover i hierarkiet. Se også neste oppgave.

Oppgave 2



Eksempler på arv:

>> arves til **ifstream**. Det vil si at operatoren kan brukes både på filobjekter og **cin**.

Alle funksjonene vist på figuren arves av klassen **fstream**. Imidlertid har denne klassen sine egne utgaver av **open( )** og **close( )** som overstyrer de som er arvet.

Oppgave 3

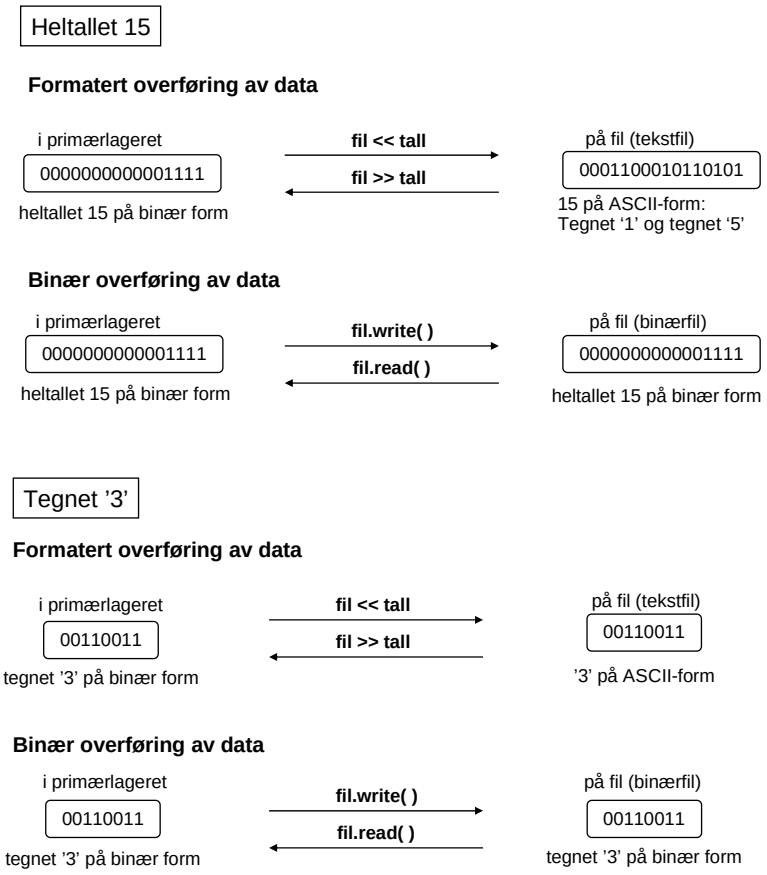
Det interne navnet til en innfil tilhører klassen **ifstream**, mens det interne navnet til en utfil tilhører klassen **ofstream**. **cin** tilhører klassen **istream**. **cout** tilhører klassen **ostream**. Funksjonen **kopierFil( )** som er definert i dette underkapitlet gjør nytte av at << arver fra **ostream** og at det aktuelle argumentet til **getline( )** kan tilhøre både klassen **istream** og **ifstream**.

Oppgave 4

cin og **cout** kan ikke være argumenter.

Kapittel 11-5

Oppgave 1



Kapittel 11-6

Oppgave 1

Fire byte kan lagre heltall med fortegn opptil 2147483647 på binær form. På en tekstfil trenger tallet -2147483647 hele 11 byte. I tillegg kommer mellomrom for å skille det fra neste tall eller tekst. Heltall på ett siffer og fortegn trenger tre byte på tekstform, det vil si mindre plass enn på binær

form. Alle heltall på to siffer eller mer (inkl. fortegn) bruker minst like mye plass på tekstform som på binærform. Flyttall kan ta mindre plass på en tekstfil enn på en binærfil. Eksempel: Tallet 2.5 bruker 4 byte på en tekstfil. I vår kompilator bruker en **double** 8 byte.

Oppgave 2

```
cout.precision(DBL_DIG);  // fra header-filen cfloat
cout << tall << endl;
```

Oppgave 3

```
&tall1
&tall2
heltall
&heltall[10]
```

Oppgave 4

```
const int n = 20;
ofstream ut;
ut.open("test1.bin", ios::binary);
ut.write((char *)&tall1, sizeof(tall1));
ut.write((char *)&tall2, sizeof(tall2));
ut.write((char *)heltall, n*sizeof(heltall[0]));
ut.write((char *)&heltall[10], sizeof(heltall[10]));
ut.close();
```

Oppgave 5

Vi regner at en **int** lbruker 4 byte, mens en **double** bruker 8 byte. Vi kan da sette opp følgende regnestykke:

tall1	4 byte
tall2	8 byte
heltall	80 byte
heltall[10]	4 byte
	96 byte

Følgende kodebit finner denne verdien (se kapittel 11-7):

```
ut.seekp(0, ios::end);   posisjoner skrivepekeren til filslutt
long int antByte = ut.tellp();   henter ut verdien til skrivepekeren
cout << "Antall byte på filen er " << antByte << endl;
```

Oppgave 6

```
#include <ctime>
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
  const int    antall = 10000;
  const double tall = 1.25365754321;
  clock_t      start;
  clock_t      slutt;

  ofstream utBinaer;
  utBinaer.open("test4.bin");
  int teller;
  start = clock();
  for (teller = 0; teller < antall; teller++) {
    utBinaer.write((char *)&tall, sizeof(tall));
    }
  slutt = clock();
  double tidBinaer = (double)(slutt - start) / CLOCKS_PER_SEC;
  utBinaer.close();

  ofstream utTekst;
  utTekst.open("test4.txt");
  start = clock();
  for (teller = 0; teller < antall; teller++) {
    utTekst << tall;
    }
  slutt = clock();
  double tidTekst = (double)(slutt - start) / CLOCKS_PER_SEC;
  utTekst.close();

  cout << "Binær overføring: " << tidBinaer << " sek\n";
  cout << "Tekst-overføring: " << tidTekst << " sek\n";
  return 0;
} // main
```

clock_t er beskrevet kort i vedlegg 6. Ved kjøring av programmet fikk vi 0.9 sek kontra 1.4 sek..

Oppgave 7

```
#include <iostream>
using namespace std;
int main() {
  // Som binærfil
  ofstream ut;
  ut.open("test.bin", ios::binary);
  int tabell[] = {10, 97, 10, 50};
  ut.write((char *)tabell, sizeof(tabell));
  cout << "Binærfilen er " << ut.tellp() << " byte lang.\n";
  ut.close();
  char tegn;
  ifstream inn;
  inn.open("test.bin", ios::binary);
```

```
  inn.read(&tegn, 1);
  while (!inn.fail()) {
    cout << (int)tegn << ' ';  //NB! "casting" til int
    inn.read(&tegn, 1);
    }
  cout << endl;
  inn.close();
  inn.clear();  // skal bruke objektet en gang til

  // Som tekstfil
  cout << "Tekstfil\n";
  ut.open("test.dat");
  ut.write((char *)tabell, sizeof(tabell));
  cout << "Tekstfilen er " << ut.tellp() << " byte lang.\n";
  ut.close();
  inn.open("test.dat", ios::binary);
  inn.read(&tegn, 1);
  while (!inn.fail()) {
    cout << (int)tegn << ' ';
    inn.read(&tegn, 1);
    }
  cout << endl;
  return 0;
} // main
```

Kjøring av programmet med Microsoft Visual C++:

```
Binærfilen er 16 byte lang.
10 0 0 0 97 0 0 0 10 0 0 0 50 0 0 0
Tekstfil
Tekstfilen er 18 byte lang.
13 10 0 0 0 97 0 0 0 13 10 0 0 0 50 0 0 0
```

I denne kompilatoren legger hvert heltall beslag på 4 byte. Tallene her er små, og bare venstre byte blir brukt. Dette er grunnen til at vi for hver tallverdi får ut tre tegn som er 0.

Dersom vi ikke åpner filen som binærfil, får vi ut tallet 13 foran hvert 10-tall. Da tror kjøresystemet at 10-tallet betyr linjeskift, og dermed skal også tallverdien 13 skrives ut, se kapittel 11-2. Merk at dette gjelder kun dersom linjeskift består av to tegn. Vi vil ikke oppleve dette på et Unix-system.

I første tilfelle er størrelsen på filen det antall byte som fire heltall trenger, dvs. 16 byte. I andre tilfelle er filen utvidet med to byte.

Kapittel 11-7

Oppgave 1

For å få til å bruke både funksjonene **tellg( )** og **tellp( )** må typen til filobjektet være **fstream**. (Dersom filtypen er **ifstream**, kan vi bruke bare **tellg( )**, og dersom den er **ofstream**, kan vi bruke bare **tellp( )**.) Filen må eksistere på forhånd, og den må åpnes med **ios::in | ios::out | ios::binary**. Første del av binut.cpp ser dermed slik ut:

```
fstream ut;
ut.open(filnavn.c_str(), ios::in | ios::out | ios::binary);

// Først en enkelt variabel:
ut.write((char *)(&tall), sizeof(tall));
cout << "Verdien til skrivepekeren er " << ut.tellp() <<
                                                    endl;
cout << "Verdien til lesepekeren er " << ut.tellg() <<
                                                    endl;


// Skriver så hele tabellen på en gang:
ut.write((char *)(tabell), antall * sizeof(tabell[0]));
cout << "Verdien til skrivepekeren er " << ut.tellp() <<
                                                     endl;
cout << "Verdien til lesepekeren er " << ut.tellg() << endl;
```

I vår kompilator får vi her følgende utskrift:

```
Verdien til skrivepekeren er 4
Verdien til lesepekeren er 4
Verdien til skrivepekeren er 24
Verdien til lesepekeren er 24
```

Vi observerer at *verdiene til de to filpekerne er de samme hele tiden*. Dette er det viktig å være klar over dersom vi trenger å holde orden på hvor langt vi til enhver tid er kommet ved skriving og lesing til en fil.

Oppgave 2

Vi må bruke binærfil, fordi det er nødvendig at hvert tall tar like stor plass, også etter at de er ganget med 10.

```
#include <fstream>
using namespace std;
int main()
{
  // Må først lage filen. Da er det greit å bruke en tabell:
  const int antall = 10;
  int tall[antall] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  ofstream ut;
```

```
  ut.open("test.bin", ios::binary);
  ut.write((char *)tall, antall * sizeof(tall[0]));
  ut.close();

  // Her kommer løsningen på oppgaven:
  fstream fil;
  fil.open("test.bin", ios::in | ios::out | ios::binary);
  int t;
  fil.read((char *)&t, sizeof(t));
  int tallnr = 0;
  while (!fil.eof()) {
    t *= 10;
    fil.seekp(tallnr * sizeof(t), ios::beg);
    fil.write((char *)&t, sizeof(t));
    fil.seekg((tallnr + 1) * sizeof(t), ios::beg);
    fil.read((char *)&t, sizeof(t));
    tallnr++;
    }
  fil.close();

  // Kontrollutskrift:
  ifstream inn;
  inn.open(filnavn.c_str(), ios::binary);
  inn.read((char *)tall, antall * sizeof(tall[0]));
  inn.close();
  for (int i = 0; i < antall; i++) cout << tall[i] << " ";
  cout << endl;

  return 0;
} // main

/* Kjøring av programmet:
10 20 30 40 50 60 70 80 90 100
*/
```

Kapittel 11-8

Oppgave 1

Begynnelsen på while-blokka skal nå se slik ut:

```
while (!cin.eof()) {
```

Kapittel 12-1

Oppgave 1

Kompilatoren starter først preprosessoren. Alle linjer som begynner med tegnet # behandles. Deretter starter selve kompileringen. Det vil si at kildekoden tolkes og oversettes til maskinspråk. Med unntak av eventuelle filer som er "inkludert" under preprosesseringen, ser ikke kompilatoren andre filer enn den som den arbeider med i øyeblikket. Det betyr at alle egendefinerte ord (funksjonsnavn, klassenavn, variabelnavn, konstantnavn, osv.) må være forklart ("deklarert") i denne filen. Og et navn må alltid deklareres før det brukes. Legg merke til at det er nok at navnet deklareres, det er ikke nødvendig å definere det.

Lenkeren syr sammen alle de definisjonene som er nødvendige for at programmet skal kunne kjøres. Lenkeren greier seg ikke med bare deklarasjoner, den må ha definisjonene.

Eksempel på en deklarasjon som ikke er en definisjon:

```
void funk(int a, int &b);
```

Deklarasjoner som også er definisjoner:

```
int a;
void skriv()
{
  cout << "Her skriver jeg et eller annet!";
} // skriv
```

En klasse, slik vi har lært å sette den opp, er en klasse*definisjon*. Men medlemsfunksjonene i klassen er bare deklarert, de defineres utenfor klassen.

Oppgave 2

Se løsningen på forrige oppgave.

Oppgave 3

En protyp er det samme som en funksjonsdeklarasjon. Se løsningen til oppgave 1.

Oppgave 4

Kompilatoren trenger ikke å kjenne mer enn grensesnittet til funksjonen. Lenkeren må imidlertid vite hva funksjonen gjør, da dette nødvendigvis er en del av det endelige programmet.

Oppgave 5

#include-setningen etterfølges av et filnavn. Preprosessoren legger innholdet i denne filen inn på den plassen der **#include**-setningen står. Kompilatoren ser dermed ikke **#include**-setningen, men i stedet det som er lagt inn på den plassen. (Den fysiske filen på disken forandres ikke.)

Kapittel 12-2

Oppgave 1

```
#include <iostream>
using namespace std;

//-------------------------------------------------------------
namespace langMaaned {
  const int antallDager = 31;
  void skrivMnd() {
    cout <<
      "januar, mars, mai, juli, august, oktober, desember";
  } // skrivMnd
} // langMaaned

//-------------------------------------------------------------
namespace kortMaaned {
  const int antallDager = 30;
  void skrivMnd() {
    cout << "april, juni, september, november";
  } // skrivMnd
} // kortMaaned

//-------------------------------------------------------------
namespace februar {
  const int antallDager = 28;
  void skrivMnd() {
    cout << "februar";
  } // skrivMnd
} // februar

//-------------------------------------------------------------

int main()
```

```
{
  {
    using namespace langMaaned;
    cout << "Disse månedene har " << antallDager << ":\n";
    skrivMnd();
    cout << endl;
  }
  {
    using namespace kortMaaned;
    cout << "Disse månedene har " << antallDager << ":\n";
    skrivMnd();
    cout << endl;
  }
  {
    using namespace februar;
    cout << "Disse månedene har " << antallDager << ":\n";
    skrivMnd();
    cout << endl;
  }

  return 0;
} // main

/* Kjøring av programmet:
Disse månedene har 31:
januar, mars, mai, juli, august, oktober, desember
Disse månedene har 30:
april, juni, september, november
Disse månedene har 28:
februar
*/
```

Kapittel 12-3

Oppgave 2

```
#define KVADRAT(x) (x) * (x)
```

Dette går bra dersom **x** ikke inngår i uttrykk. **x** kan selv være et uttrykk. Eksempel på uttrykk der dette ikke går bra: **10.0 / KVADRAT(2.0 + 3.0)** blir byttet ut med **10.0 / (2.0 + 3.0) * (2.0 + 3.0)**. Verdien til uttrykket blir (10.0 / 5.0) * 5.0 = 10.0. Vi hadde antakelig ment (10.0/ (5.0 * 5.0)) = 0.4.

```
#define KVADRAT (x)  x * x
```

Her kan generelt ikke **x** være et utrykk: **KVADRAT(2 + 3)** blir byttet ut med 2 + 3 * 2 + 3, som er lik 2 + (3 * 2) + 3 = 9. Vi hadde antakelig ment (2+ 3) * (2 + 3) = 25.

Kapittel 13-1

Oppgave 1

Korrekt setning med betingelsesoperatoren:

```
svar = operator == '+' ? a + b : a - b;
```

Som **if-else**-setning:

```
if (operator == '+') svar = a + b;
else svar = a - b;
```

Oppgave 2

Funksjonen returnerer verdien av kommauttrykket **a + b, a - b** dvs. **a - b**. Summen av **a** og **b** returneres ikke.

Kapittel 13-2

Oppgave 1

```
switch (dag) {
  case man:  Komma mellom man og tirs gir et uttrykk
             der verdien ikke er kjent under kompilering.
    // ikke break
  case tirs:
    cout << "Begynnelsen av uka\n";
    break;       Husk break!
  case ons:
    // ikke break
  case tors:
    cout << "Midten av uka\n";
    break;
  case fre:
    cout << "Slutten av uka\n";
    break;
  case loer:
    // ikke break
  case soen:
    cout << "Helg\n";
    break;
  default:       Husk default med break!
    break;
  } // switch    Ikke semikolon etter }
```

Kapittel 13-3

Oppgave 1

```
for(int tall = 1, sum = 0; tall <= 10; sum += tall, tall++);
cout << sum << endl;
```

Oppgave 2

```
for (char bokstav = 'A';
     bokstav <= 'Z';
     cout << bokstav << " har ASCII-verdi: " <<
                          (int)bokstav << endl,
       bokstav++);
```

Oppgave 3

```
const int antlinjer = 2;
const int antkolonner = 3;
int tabell[antlinjer][antkolonner] = {1, 2, 3, 4, 5, 6};
for (int sum2 = 0, linjenr = 0; linjenr < antlinjer;
                                  linjenr++)
  for (int kolonnenr = 0;
    kolonnenr < antkolonner;
    sum2 += tabell[linjenr][kolonnenr], kolonnenr++);
  cout << sum2 << endl;
```

Oppgave 4

```
for (int antblanke = 3, antstjerner = 1;
     antstjerner <= 4;
     antstjerner++, antblanke--, cout << endl) {
  for (int blankNr = 0;
       blankNr < antblanke;
       cout << " ", blankNr++);
  for (int stjerneNr = 0;
       stjerneNr < antstjerner;
       cout << "* ", stjerneNr++);
  }
```

Kapittel 13-5

Oppgave 1

a) **for**-løkke

b) **do-while**-løkke

c) **for**-løkke

d) **do-while**-løkke

Kapittel 14-1

Oppgave 1

Det er kun funksjonshodene som må forandres:

```
const string &Navn::finnEtternavn() const ...
void Navn::settEtternavn(const string &nyttEtternavn) ...
```

Oppgave 2

***const** foran og etter funksjonsdeklarasjonen:*

```
const string &finnNavn(void) const;
```

Den første **const**'en betyr at det som funksjonen returnerer må legges i en konstant, her en **const-**referanse, eventuelt kopieres over til et ikke-**const** objekt. Det sikrer at klienten ikke kan oppdatere det datamedlemmet som referansen referer til. Den andre **const**'en betyr at et kall på medlemsfunksjonen ikke forandrer på objektet den kalles på vegne av.

***const** foran et funksjonsargument:*

```
void settNavn(const string &nyttNavn);
```

const betyr at funksjonen ikke forandrer på det aktuelle argumentet.

Oppgave 3

```
#include <string>
using namespace std;

const int maksAntRuter = 20; // hver vei

 class Bondesjakk {
   public:
     enum verdier {  // aktuelle verdier i ei rute
       blank, kryss, sirkel};

     void nyttSpill( // blanker ut alle rutene
```

```
        const string &spillerKryss,   // Inn, spillerNavn grunn b)
        const string &spillerSirkel); // Inn, spillerNavn grunn b)

    // Følgende funksjon returnerer false hvis ugyldig
    // indeks eller spillernavn. Om argumentene:
    // spillerNavn bestemmer kryss eller sirkel
    // ok er true dersom korrekt spiller (se data-
    // medlemmet kryssSisteSpiller)
    // seier er true dersom siste trekk medførte seier
    bool settMarkering(
            const string &spillerNavn,     // Inn grunn b)
             int           indeksVannrett, // Inn
             int           indeksLoddrett, // Inn
             bool         &ok,             // Ut  grunn a)
           bool         &seier);           // Ut   grunn a)

     const string &finnSpillerKryss() const;      grunn b) og c)
     const string &finnSpillerSirkel() const;     grunn b) og c)
     void   finnSpillere(
               string &spillerKryss,        // Ut   grunn a)
               string &spillerSirkel) const;// Ut   grunn a) og c)
     int    finnAntallSpilleomganger() const;     grunn c)
     void   tegnSpilleskjema() const;  //          grunn c)

   private:
     string  spillerKryss;
     string  spillerSirkel;
     verdier ruteNett[maksAntRuter][maksAntRuter];

    // Følgende datamedlem oppdateres for hvert trekk.
    // Den skal sikre at spillerne foretar annnenhvert
    // trekk.
     bool    kryssSisteSpiller; // true hvis siste spiller spillerKryss
}; // Bondesjakk
```

Kapittel 14-2

Oppgave 1

a) Feil

b) Riktig

c) Feil

d) Feil

e) Feil

f) Første setning er riktig, andre setning er feil. Standardverdiene skal settes i prototypen, ikke i definisjonen.

g) Riktig

h) Riktig

i) Riktig (egendefinert typeomforming)

j) Feil

k) Riktig

l) Riktig

m) Riktig

n) Feil

Oppgave 2a

```
Saldo: 700
Kredittgrense: 500
Saldo: 700, kredittgrense: 500
Saldo: 0, kredittgrense: 3000
Hurra!
```

Oppgave 2b

```
Konto kontoA;  // konstruktøren med std.verdier på argumentene
lesData(kontoA);  // ingen konstruktør (referanseoverføring)
Konto kontoB(3000); // konstr. med std.verdi på 2.argument
skrivData(kontoA);  // standard kopikonstruktør
skrivData(kontoB);  // standard kopikonstruktør

Konto kontoC = kontoB; // standard kopikonstruktør
if (kontoC.finnGrense() > 1000) cout << "Hurra!\n";
else cout << "Ikke hurra!\n";
```

Kapittel 14-4

Nedenfor er implementasjonen av de nye konstruktørene og funksjonene vist. I tillegg må prototypene skrives opp inne i de klassene de tilhører.

Klientprogrammet for alle funksjonene er tatt med helt til slutt.

Oppgave 1

```
Prosjektoppgave::Prosjektoppgave(int initNr,
 const string &initTittel,
 const string &initFornavnStud,
 const string &initMellomnavnStud,
 const string &initEtternavnStud,
 const string &initFornavnVeil,
 const string &initMellomnavnVeil,
 const string &initEtternavnVeil)
: oppgaveNr(initNr),
  tittel(initTittel),
  student(initFornavnStud, initMellomnavnStud,
          initEtternavnStud),
  veileder(initFornavnVeil, initMellomnavnVeil,
           initEtternavnVeil)
{}

Student::Student(const string &initFornavn,
  const string &initMellomnavn,
  const string &initEtternavn)
: navn(initFornavn, initMellomnavn, initEtternavn)
{}
```

Oppgave 2

```
void Student::settEtternavn(const string &nyttEtternavn)
{
  navn.settEtternavn(nyttEtternavn);
}  //settEtternavn

void Prosjektoppgave::settEtternavnStudent(
                        const string &nyttEtternavnStud)
{
  student.settEtternavn(nyttEtternavnStud);
} // settEtternavnStudent
```

Oppgave 3

```
const string &Prosjektoppgave::finnFornavnVeileder() const
{
  return veileder.finnFornavn();
} // finnFornavnVeileder
```

Oppgave 4

```
int Prosjektoppgave::sammenliknKarakter(
            const Prosjektoppgave &denAndre) const
{
  char denneKarakter = finnKarakter();
  char denAndreKarakter = denAndre.finnKarakter();
  if (denneKarakter < denAndreKarakter) return -1;
  else if (denneKarakter > denAndreKarakter) return 1;
  else return 0;
} // sammenliknKarakter
```

Felles klientprogram for alle fire oppgavene

```
int main()
{

  // Oppgave 1
  Prosjektoppgave oppgave(100, "Oppgave X", "Ole",
    "Petter", "Hansen", "Per", "Ivar", "Ås");

  // Skriver ut alle data ved å bruke finn-funksjoner
  cout <<"Oppgave nr " << oppgave.finnNr() << " " <<
    oppgave.finnTittel() << "\n" << "Student: " <<
    oppgave.finnStudnavn().finnNavn() << "\n" <<
    "Veileder: " <<
    oppgave.finnVeiledernavn().finnNavn() << endl;

  // Oppgave 2
  oppgave.settEtternavnStudent("Heia");
  cout << "Nytt studentnavn: " <<
          oppgave.finnStudnavn().finnNavn() << endl;

  // Oppgave 3
  cout << "Fornavnet til veileder er " <<
          oppgave.finnFornavnVeileder() << endl;

  // Oppgave 4
  // Lager først to oppgaver til:
  Prosjektoppgave oppgave2(101, "Oppgave Y", "Arne Lillebo",
                           "Ingrid Jensen");
  Prosjektoppgave oppgave3(102, "Oppgave Z", "Eva Hansen",
                         "Ingrid Jensen");

  oppgave.settKarakter('A');
  oppgave2.settKarakter('C');
  oppgave3.settKarakter('C');

  // Oppgave 4
  // skal få utskriftene -1 0 1
  cout << oppgave.sammenliknKarakter(oppgave2) << endl;
  cout << oppgave2.sammenliknKarakter(oppgave3) << endl;
  cout << oppgave2.sammenliknKarakter(oppgave) << endl;

  return 0;
} // main
```

```
/* Utskrift:
Oppgave nr 100 Oppgave X
Student: Ole Petter Hansen
Veileder: Per Ivar +s
Nytt studentnavn: Ole Petter Heia
Fornavnet til veileder er Per
-1
0
1
*/
```

Kapittel 15-1

Oppgave 1

```
bool Navn::operator==(const Navn &detAndre) const
{
  return sammenlikn(detAndre) == 0;
} // operator==
```

Oppgave 2

```
class Navn {
  public:
    …
    bool operator&&(const Navn &detAndre) const;
    bool operator!() const;
    char operator[](int nr) const;
    …
  }; // Navn
```

Mange andre mulige prototyper fins.

Kapittel 15-2

Oppgave 1

```
Broek Broek::operator+() const
{
  return *this;  // returnerer en kopi av *this
} // operator+
```

Oppgave 2

hjelp er en lokal variabel som vil forsvinne når funksjonen er ferdig. Derfor kan vi ikke returnere en referanse til denne variabelen, bare en kopi av den. ***this** vil derimot ikke forsvinne, så den kan vi returnere en referanse til.

Oppgave 3

a = b = c; tolkes som **a = (b = c);** Verdien av siste uttrykk er **const**, OK.

(a = b) = c; gir et **const**-uttrykk på venstre side, kompileringsfeil.

++(a = b); siden vi ikke kan øke verdien av et **const**-uttrykk, får vi kompileringsfeil.

Kapittel 15-3

Oppgave 1

```
Broek Broek::operator-(int heltall) const
{
  Broek hjelp(-heltall, 1);
  hjelp += *this;
  return hjelp;
} // operator-

//------------------------------------------------------------
Broek operator-(int heltall, const Broek &denAndre)
{
  Broek hjelp(heltall, 1);
  hjelp -= denAndre;
  return hjelp;
} // operator-
```

Oppgave 2

Først blir 5 + 3 regnet ut. Den vanlige betydingen av + mellom heltall brukes.Resultatet blir et heltall som skal adderes til brøken **broekA**. Da blir ikke-medlemsfunksjonen **Broek operator+(int heltall, const Broek &denAndre)** brukt, og resultatet blir en brøk. Deretter blir medlemsfunksjonen **Broek Broek::operator+(int heltall) const** brukt. slik at vi får addert til heltallet 7. Til slutt blir medlemsfunksjonen **Broek Broek::operator+(const Broek &denAndre) const** brukt.

Oppgave 3

```
ostream &Broek::operator<<(ostream &ut)
{
  ut << teller << " / " << nevner;
  return ut;
} // operator<<

//-------------------------------------------------------------
// Funksjonskall
Broek a;
…
a << cout << endl; // OK
cout << "Svaret blir " << a; // Kompileringsfeil (hvis ikke den
vanlige også eksisterer)
```

Ved sammenkjeding må brøken stå først. Det er lite hensiktsmessig siden vi ofte vil ønske å skrive ut andre ting (for eksempel tekst) foran. En annen grunn til at vi vil ha argumentene i motsatt retning er at det er lettest å huske (og forstå) hvis samme operator har argumentene i samme rekkefølge i alle versjonene.

Oppgave 4

```
enum dag {
  mandag, tirsdag, onsdag, torsdag, fredag, loerdag, soendag};
const string dagnavn[] = {"mandag", "tirsdag", "onsdag",
  "torsdag", "fredag", "lørdag", "søndag"};

ostream &operator<<(ostream &ut, dag dagen)
{
  ut << dagnavn[dag];
  return ut;
} // operator<<
```

Oppgave 5

Når addisjonen ikke er definert direkte, forsøkes typeomforming for å få det til. Siden vi nå har en konstruktør som kan omforme fra et heltall til en **Broek**, utføres denne omformingen slik at addisjonen blir definert.

Hvis vi skriver uttrykket i motsatt rekkefølge, går det ikke automatisk. Men vi kan "caste" slik at det fungerer: **(Broek)3 + broekA** vil gå bra.

Kapittel 15-4

Oppgave 1

Fordi de da ville få nøyaktig samme prototyp som prefiksversjonene.

Oppgave 2

```
Broek Broek::operator()(int potens) const
{
  Broek hjelp(1, 1)
  for (int i = 0; i < potens; i++) {
    hjelp *= *this;
    }
  return hjelp;
} // operator()
```

Oppgave 3

Nei, det anbefales ikke. Det er for lite åpenbart hva skrivemåten betyr.

Kapittel 16-1

Oppgave 1

```
int a = 5:
int &b;  // må initieres. int &b = a;
int *c;
c = &b;
*a = *b + *c; // a og b er ikke pekere. a = b + *c;
&b = 2; // adressen til en variabel kan ikke endres. b = 2;
```

Oppgave 2

a) **double *peker = &tall**;

b) **double &ref = tall**;

c) **tall = 5.0;**
***peker = 5.0;**
ref = 5.0;

Kapittel 16-2

Oppgave 1

```
int finnSum(
  const int *tabell,    // Inn
  int        aktLengde) // Inn
{
  int sum = 0;
  for (int i = 0; i < aktLengde; i++) {
    sum += tabell[i];
    }
  return sum;
} // finnSum

….

const int maks = 50;
int tabell[maks];

a)  int sum1 = finnSum(tabell, 10);
b)  int sum2 = finnSum(&tabell[10], 15);
c)  int sum3 = finnSum(&tabell[maks – 30], 30);
```

Oppgave 2

```
int finnMaks(cosnt int *tabell, int antall)
{
  const int *peker // siden tabell er konstantpeker
            = tabell;  // tabell er en peker
  int teller;
  for (teller = 0; teller < antall; teller++) {
    if (tabell[teller] > *peker) {
      peker = &tabell[teller]; // pekeren skal flyttes, vi skal
                               // ikke endre det den peker til.
    }
  }
  return *peker; // skal finne verdien av det pekeren peker
                 // til, ikke adressen til pekeren.
} // finnMaks
```

Kapittel 16-3

Oppgave 1

```
bool kopier(  // false hvis for få elementer i kopien
  const int *original,       // Inn
  int        lengdeOriginal, // Inn, aktuell lengde
  int        lengdeKopi,     // Inn, maksimal lengde
```

```
  int       *kopi)           // Ut
{
  if (lengdeOriginal <= lengdeKopi) {
    for (int nr = 0; nr < lengdeOriginal; nr++) {
      *(kopi + nr) = *(original + nr);
      }
    return true;
    }
  else return false;
} // kopier
```

Alternativt:

```
{
  if …
    const int *oPeker = original;
    int       *kPeker = kopi;
    for…
      *kPeker = *oPeker;
      kPeker++;
      oPeker++;
```

Oppgave 2

Funksjonen kopierer tabellen fra **tab1** til **tab2**, men verdiene blir lagt i motsatt rekkefølge.

Med ***peker = *tab1 – 1**: Vi ville fått tilordningene

```
tab2[antall – 1] = tab1[1] - 1;
…
tab2[0] = tab1[antall] – 1;
```

Kapittel 16-4

Oppgave 1

Hvis det vi leter etter ikke fins, fortsetter vi videre forbi det dataområdet som er avsatt til tabellen slik at vi får ødelagt andre data. Uforutsigelige ting kan skje hvis vi kjører programmet.

Kapittel 16-5

Oppgave 1

```
Konto konto;
Konto *kontopeker = &konto;
konto.settSaldo(1000.0);
konto.settGrense(0.0);
double grense = kontopeker->finnGrense();
double saldo = kontopeker->finnSaldo();
```

Oppgave 2

```
const int maks = 10;
Konto     tabell[maks];
Konto    *peker = tabell;
for (int nr = 0; nr < maks; nr++) {
  peker->settGrense(0.0);
  peker++;
  }
```

Kapittel 16-6

Oppgave 1

```
int antall;
cout << "Hvor mange kontoer har du?";
cin >> antall;

Konto *kontoene = new Konto[antall];

delete [] kontoene;
```

Oppgave 2

```
void utvidTab(
  int *&tabell,       // Inn/ut
  int  &maksLengde)   // Inn/ut
{
  int *nyTab = new int[2 * maksLengde];
  for (int i=0; i<maksLengde; i++) {
    nyTab[i] = tabell[i];
    }
  delete [] tabell;
  tabell = nyTab;
  maksLengde *= 2;
} // utvidTab
```

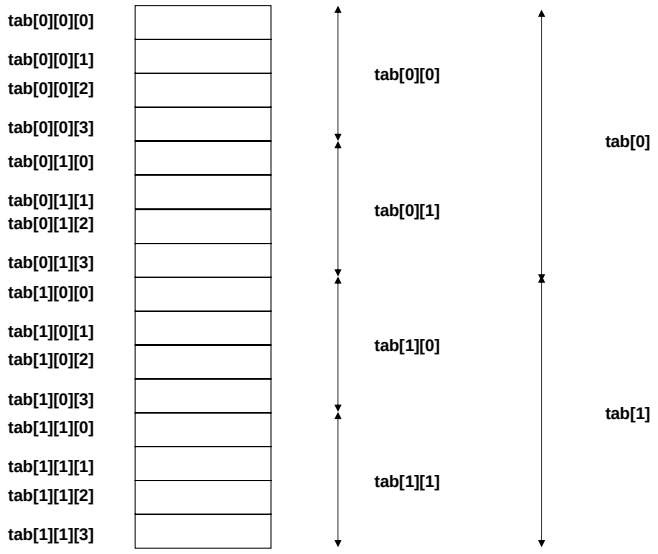
Kapittel 16-7

Oppgave 1

```
double gjSnitt(
```

```
  const double *tabell,    // Inn
  int           antall)    // Inn
{
  double sum = 0;
  for (int nr = 0; nr < antall; nr++) {
    sum += tabell[nr];
    }
  if (antall > 0) return sum / antall;
  else return 0.0;
} // gjSnitt
//-------------------------------------------------------
const int antLin = 10;
const int antKol = 20;
int main()
{
  double tabell[antLin][antKol];
  // gi verdi til elementene
  double snitt = gjSnitt(tabell[0], antLin * antKol); // a)
  snitt = gjSnitt(tabell[5], antKol);  // b)
  // oppgave c) er ikke mulig
  return 0;
}
```

Oppgave 2



Oppgave 3

```
const int antall = 10;
```

```
int tabell[antall][antall];

nullEnKolonne(antall, antall + 1, 0, tabell[0]);
```

Oppgave 4

Funksjonen snur om på tabellen slik at linjer blir kolonner og motsatt.

```
funk(5, 8, b[0], c[0]);
```

Kolonnene 0 til 4 i **c** blir lik linjene 0 til 4 i **b**. Kolonnene 5 til 7 i **c** forandres ikke.

```
funk(3, 3, b[1], &c[3][2]);
```

Funksjonen betrakter tabellene som 3x3-tabeller med start i henholdsvis **b[1][0]** og **c[3][2]**. Får følgende tilordninger: **c[3][2] = b[1][0]**, **c[3][5] = b[1][1]**, **c[4][0] = b[1][2]**, **c[3][3] = b[1][3]**, **c[3][6] = b[1][4]**, **c[4][1] = b[1][5]**, **c[3][4] = b[1][6]**, **c[3][7] = b[1][7]**, **c[4][2] = b[2][0]**

```
funk(10, 10, a[0], a[0]);
```

Første kolonne blir lik første linje osv. Men siden vi bruker samme tabellen både som **tab1** og som **tab2**, vil de forandringene som etter hvert skjer i kolonnene, bli med når linja brukes.

```
funk(8, 8, c[0], b[0]);
```

Også her snur vi linjer og kolonner. Men vi går utenfor grensene til **b**, slik at vi ødelegger andre data.

Kapittel 16-8

Oppgave 1

```
*(*(pekerTab + linje) + kol)
```

Oppgave 2

```
const char maks = 10;
string strengtabell[maks];
char *pekere[maks];
int   antall;
```

```
// gi verdi til (noen av) elementene i strengtabell
for (int i=0; i < antall; i++) {
  int pos = strengtabell[i].length() – 1;
  pekere[i] = &strengtabell[i][pos];
  }
```

Kapittel 17-2

Oppgave 1

```
Størrelse: 3, kapasitet: 4
Utskrift A:
4/7 3/4 2/5
Utskrift B:
7/8 4/7 3/4
Utskrift C:
3/4 2/5 4/7
```

Oppgave 2

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace  std;

int main()
{
  // Oppgave b)
  vector<double> tall;
  tall.push_back(3.15);
  cout << "Etter 1.tall: Size = " << tall.size() <<
          ", kapasitet: " << tall.capacity() << endl;
  tall.push_back(-5.3);
  cout << "Etter 2.tall: Size = " << tall.size() <<
          ", kapasitet: " << tall.capacity() << endl;
  tall.push_back(-4.2);
  cout << "Etter 3.tall: Size = " << tall.size() <<
          ", kapasitet: " << tall.capacity() << endl;
  tall.push_back(17.6);
  cout << "Etter 4.tall: Size = " << tall.size() <<
          ", kapasitet: " << tall.capacity() << endl;
  tall.push_back(23);
  cout << "Etter 5.tall: Size = " << tall.size() <<
          ", kapasitet: " << tall.capacity() << endl;

  // Oppgave a)
  if (!tall.empty()) cout << "Første element er " <<
       tall.front() << endl;
  if (!tall.empty()) cout << "Siste element er " <<
       tall.back() << endl;

  // Oppgave c)
  vector<double> tall2(5, 2.5);
```

```
  cout << "Vektor tall2: ";
  for (int i = 0; i < tall2.size(); i++) {
    cout << tall2[i] << " ";
    }
  cout << endl << endl;

  // Oppgave d)
  tall.swap(tall2);
  cout << "Vektor tall2 etter swapping: ";
  for (int j = 0; j < tall2.size(); j++) {
    cout << tall2[j] << " ";
    }
  cout << endl;

  cout << "Vektor tall etter swapping: ";
  for (int k = 0; k < tall.size(); k++) cout << tall[k] << " ";
  cout << endl;

  // Oppgave e)
  tall.clear();
  tall2.clear();
  cout << "Etter tømming, tall: Size = " << tall.size() <<
          ", kapasitet: " << tall.capacity() << endl;
  cout << "Etter tømming, tall2: Size = " << tall2.size() <<
          ", kapasitet: " << tall2.capacity() << endl;

  return 0;
} // main

/* Utskrift:
Etter 1.tall: Size = 1, kapasitet: 1
Etter 2.tall: Size = 2, kapasitet: 2
Etter 3.tall: Size = 3, kapasitet: 4
Etter 4.tall: Size = 4, kapasitet: 4
Etter 5.tall: Size = 5, kapasitet: 8
Første element er 3.15
Siste element er 23
Vektor tall2: 2.5 2.5 2.5 2.5 2.5

Vektor tall2 etter swapping: 3.15 -5.3 -4.2 17.6 23
Vektor tall etter swapping: 2.5 2.5 2.5 2.5 2.5
Etter tømming, tall: Size = 0, kapasitet: 5
Etter tømming, tall2: Size = 0, kapasitet: 8
*/
```

Kapitte 17-3

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "elev.h"

int main()
{
  vector<Elev> elevene;
  elevene.push_back(Elev("Berit Hansen"));
  elevene.push_back(Elev("Ingrid Jensen"));
  elevene.push_back(Elev("Anders Moe"));
```

```
  elevene.push_back(Elev("Hanne Andersen"));
  elevene.push_back(Elev("Ingrid Havstad"));

  // Sorterer elevene slik det er gitt av operatoren <
  // i klassen Elev
  sort(elevene.begin(), elevene.end());

  // Skriver ut data om elevene
  for (int i = 0; i < elevene.size(); i++) {
    cout << elevene[i] << endl;
    }

  cout << "\nOppgave a\n";
  vector<Elev>::reverse_iterator revIt = elevene.rbegin();
  while (revIt != elevene.rend()) { // begynnelsen
    cout << revIt->finnNavn()  << endl;
    revIt++;  // flytter oss mot begynnelsen av vektoren
    }

  cout << "\nOppgave b\n"; // sletter i alt tre elementer
  vector<Elev>::const_iterator retur =
              elevene.erase(elevene.end() - 1); // siste
  if (retur == elevene.end()) {
    cout << "Peker til elevene.end()\n";
    }
  retur = elevene.erase(elevene.begin() + 1,
                        elevene.begin() + 3);
  cout << "Peker til elementet " << retur->finnNavn() << endl;

  cout << "\nOppgave c\n";
  cout << "Size før tømming er " << elevene.size() << endl;
  cout << "Kapasitet før tømming er " <<
          elevene.capacity() << endl;
  elevene.clear();
  cout << "Size etter tømming er " << elevene.size() << endl;
  cout << "Kapasitet etter tømming er " <<
          elevene.capacity() << endl;

  return 0;
} // main

/* Utskrift
Anders Moe har 0 dager fravær.
Berit Hansen har 0 dager fravær.
Hanne Andersen har 0 dager fravær.
Ingrid Havstad har 0 dager fravær.
Ingrid Jensen har 0 dager fravær.

Oppgave a
Ingrid Jensen
Ingrid Havstad
Hanne Andersen
Berit Hansen
Anders Moe

Oppgave b
Peker til elevene.end()
```

```
Peker til elementet Ingrid Havstad

Oppgave c
Size før tømming er 2
Kapasitet før tømming er 8
Size etter tømming er 0
Kapasitet etter tømming er 8
*/
```

Kapittel 17-4

Nullstilling av fravær som standard settes opp i prototypen i klassedefinisjonen:

```
bool settFravaer(const string &navn, int nyttFravaer = 0);
```

Her følger funksjonsdefinisjonene:

```
bool Elevregister::settFravaer(const string &navn,
                               int nyttFravaer)
{
  Elev elev(navn);
  vector<Elev>::iterator eleven =
              find(elever.begin(), elever.end(), elev);
  if (eleven < elever.end()) {
    eleven->settFravaer(nyttFravaer);
    return true;
    }
  else return false;
} // settFravaer

bool Elevregister::endreNavn(const string &gmlNavn,
                             const string &nyttNavn)
{
  Elev elev(gmlNavn);
  vector<Elev>::iterator eleven =
             find(elever.begin(), elever.end(), elev);
  if (eleven < elever.end()) {
    eleven->settNavn(nyttNavn);
    return true;
    }
  else return false;
} // endreNavn

bool Elevregister::slettElev(const string &navn)
{
  Elev elev(navn);
  vector<Elev>::iterator eleven =
            find(elever.begin(), elever.end(), elev);
  if (eleven < elever.end()) {
    elever.erase(eleven);
    return true;
    }
  else return false;
} // slettElev
```

Revidert klientprogram:

```
#include <iostream>
#include "elevregister.h"
#include "nytte.h"

const string menytekst =
                  "N: Ny elev.\n"
                  "E: Endre fraværet for en elev.\n"
                  "F: Sett nytt fravær for en elev.\n"
                  "V: Nytt navn på en elev.\n"
                  "L: Slett en elev.\n"
                  "S: Skriv ut oversikt over alle elevene.\n"
                  "A: Avslutt.\n";
const string lovligValg = "NEFVLSA";
enum valg {nyElev, endreFravaer, nyttFravaer, nyttNavn, slett,
skrivUt, avslutt};

void utfoerValg(valg valget, Elevregister &registeret);

int main()
{
  Elevregister fravaeret;
  valg valget = (valg) lesMenyvalg(menytekst, lovligValg);
  while (valget != avslutt) {
    utfoerValg(valget, fravaeret);
    valget = (valg)lesMenyvalg(menytekst, lovligValg);
  } // while
  return 0;
} // main

//-------------------------------------------------------------
//
// Vedlikeholder fraværsregisteret.
//
void utfoerValg(
  valg     valget,        // Inn
  Elevregister &registeret)       // Inn/ut
{
  switch (valget) {
    case nyElev: {
      cout << "Oppgi navn: ";
      string navn = lesLinje(cin);  // fra nytte.h
      registeret.nyElev(navn);
      cout << "Ny elev registrert.\n";
      } // case
      break;
    case endreFravaer: {
      cout << "Elevens navn: ";
      string navn = lesLinje(cin);
      int dager;
      cout << "Oppgi endring i fraværet: ";
      cin >> dager;
      if (registeret.endreFravaer(navn, dager)) {
        cout << "Ok. Nå er fraværet for denne eleven " <<
                registeret.finnFravaer(navn) << " dager.\n";
        }
```

```
      else cout << "Ingen elev med dette navnet.\n";
      } // case
      break;
    case nyttFravaer: {
      cout << "Elevens navn: ";
      string navn = lesLinje(cin);
      int dager;
      cout << "Oppgi det nye fraværet: ";
      cin >> dager;
      if (registeret.settFravaer(navn, dager)) {
        cout << "Ok. Nå er fraværet for denne eleven " <<
                registeret.finnFravaer(navn) << " dager.\n";
        }
      else cout << "Ingen elev med dette navnet.\n";
      } // case
      break;
    case nyttNavn: {
      cout << "Elevens gamle navn: ";
      string navn1 = lesLinje(cin);
      cout << "Elevens nye navn: ";
      string navn2 = lesLinje(cin);
      if (registeret.endreNavn(navn1, navn2)) {
        cout << "Ok. Nå er navnet endret.\n";
        }
      else cout << "Ingen elev med dette navnet.\n";
      } // case
      break;
    case slett: {
      cout << "Elevens navn: ";
      string navn = lesLinje(cin);
      if (registeret.slettElev(navn)) {
        cout << "Ok. Nå er eleven slettet.\n";
        }
      else cout << "Ingen elev med dette navnet.\n";
      } // case
      break;
    case skrivUt: {
      cout << "\nAlle elevene:\n";
      for (int indeks = 0; indeks < registeret.finnAntElever();
           indeks++) {
        const Elev *eleven = registeret.finnElev(indeks);
        cout << "Elev nr: " << (indeks + 1) << ", " <<
                eleven->finnNavn() << " har " <<
                eleven->finnFravaer() << " dager fravær.\n";
      } // for
      cout << endl;
      } // case
      break;
    default:
      break;
    } // switch
} // utfoerValg
```

Kapittel 17-5

Oppgave 1

```
int main()
{
  vector<int> v1 = lagVektor1();
  vector<int> v2 = lagVektor2();
  skrivUt("Vektor 1: " , v1);
  skrivUt("Vektor 2: " , v2);

  cout << "\nOppgave a:\n";
  vector<int> v3(v1.size() + v2.size());
  merge(v1.begin(), v1.end(), v2.begin(), v2.end(),
        v3.begin());
  skrivUt("V3: ", v3);

  cout << "\nOppgave b:\n";

  // en vektor med ett element med verdien 12:
  vector<int> verdi(1, 12); //
  vector<int>::const_iterator it =
    find_end(v3.begin(), v3.end(), verdi.begin(), verdi.end());
  cout << "Indeks til siste forekomst av " <<
          *it << " er " << (it - v3.begin()) << endl;

  cout << "\nOppgave c:\n";
  cout << "Antall 3-tall i v3 er " <<
           count(v3.begin(), v3.end(), 3) << endl;

  cout << "\nOppgave d:\n";
  vector<int> v3kopi(v3); // tar vare på en kopi
  v3.resize(v3.size() + 5);
  copy(v3.end() - 15, v3.end() - 5,
                      v3.end() - 10);  // kan ikke brukes
  skrivUt("Ny v3 etter copy():", v3);

  swap(v3, v3kopi); // Nå er v3 som før
  v3.resize(v3.size() + 5);
  copy_backward(v3.end() - 15, v3.end() - 5, v3.end());
  skrivUt("Ny v3 etter copy_backward():", v3);

  cout << "\nOppgave e:\n";
  int gmlStr = v1.size();
  v1.resize(v1.size() + 20);
  fill_n(v1.begin() + gmlStr, 20, 4);
  cout << "Etter fill_n(), størrelse = " << v1.size() <<
          ", kapasitet: " << v1.capacity() << endl;
  skrivUt("V1:", v1);

  remove(v1.end() - 20, v1.end(), 4);
  cout << "\nEtter remove(), størrelse = " << v1.size() <<
          ", kapasitet: " << v1.capacity() << endl;
  skrivUt("V1:", v1);

  v1.erase(v1.end() - 20, v1.end());
```

```
  cout << "\nEtter erase(), størrelse = " << v1.size() <<
          ", kapasitet: " << v1.capacity() << endl;
  skrivUt("V1:", v1);

  // Kan ikke redusere kapasiteten, men kan fjerne
  // overflødig minne slik:
  vector<int> temp(v1.begin(), v1.end());
  swap(temp, v1);
  cout << "\nEtter swap(), størrelse = " << v1.size() <<
          ", kapasitet: " << v1.capacity() << endl;
  skrivUt("V1:", v1);

  return 0;
} // main

/* Utskrift
Vektor 1:
3 3 12 14 17 25 30
Vektor 2:
2 3 12 14 24

Oppgave a:
V3:
2 3 3 3 12 12 14 14 17 24 25 30

Oppgave b:
Indeks til siste forekomst av 12 er 5

Oppgave c:
Antall 3-tall i v3 er 3

Oppgave d:
Ny v3 etter copy():
2 3 3 3 12 12 14 3 3 12 12 14 3 3 12 12 14
Ny v3 etter copy_backward():
2 3 3 3 12 12 14 3 3 12 12 14 14 17 24 25 30

Oppgave e:
Etter fill_n(), størrelse = 27, kapasitet: 27
V1:
3 3 12 14 17 25 30 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

Etter remove(), størrelse = 27, kapasitet: 27
V1:
3 3 12 14 17 25 30 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

Etter erase(), størrelse = 7, kapasitet: 27
V1:
3 3 12 14 17 25 30

Etter swap(), størrelse = 7, kapasitet: 7
V1:
3 3 12 14 17 25 30
*/
```

Oppgave 2

```
// Funksjoner som brukes
bool oppgA(int tall)
{
  return (tall > 15);
} // oppgA

bool oppgB(int verdi1, int verdi2)
{
  return (abs(verdi1 - verdi2) <= 2);
} // oppgB

bool oppgC(int tall)
{
  return (tall % 2 != 0);
} // oppgC


int main()
{
  vector<int> v1 = lagVektor1();
  vector<int> v2 = lagVektor2();
  skrivUt("Vektor 1: " , v1);
  skrivUt("Vektor 2: " , v2);

  cout << "\nOppgave a:\n";
  vector<int>::iterator it =
            find_if(v1.begin(), v1.end(), oppgA);
  cout << "Første element større enn 15 er på indeks "
       << (it - v1.begin()) <<
          " og har verdien " << *it << endl;

  cout << "\nOppgave b:\n";
  bool likhet = equal(v1.begin(),
                v1.begin() + 5, v2.begin(), oppgB);
  if (likhet) cout << "1: Likhet.\n";
  else cout << "1: Ikke likhet.\n";
  likhet = equal(v1.begin(),
                 v1.begin() + 4, v2.begin(), oppgB);
  if (likhet) cout << "2: Likhet.\n";
  else cout << "2: Ikke likhet.\n";

  cout << "\nOppgave c:\n";
  replace_copy_if(v1.begin(), v1.end(),
                  v1.begin(), oppgC, 100);
  skrivUt("V1: ", v1);

  return 0;
} // main

/* Utskrift:
Vektor 1:
3 3 12 14 17 25 30
```

```
Vektor 2:
2 3 12 14 24

Oppgave a:
Første element større enn 15 er pÅ indeks 4 og har verdien 17

Oppgave b:
1: Ikke likhet.
2: Likhet.

Oppgave c:
V1:
100 100 12 14 100 100 30
*/
```

Oppgave 3

```
// Funksjoner som trengs - se implementasjon nederst
char tilStor(char tegn);
char tilLiten(char tegn);
string tilStore(string tekst);
string tilSmaa(string tekst);
bool mindreEnnChar(char c1, char c2);
bool mindreEnnString(const string &s1, const string &s2);
bool like(char c1, char c2);
bool likhet(const string &s1, const string &s2);

int main()
{
  int i;
  vector<string> navn;
  navn.push_back("Æsop");
  navn.push_back("Einar");
  navn.push_back("Åsmund");
  navn.push_back("Øyvind");

  cout << "Oppgave b:\n";
  vector<string> navnStore(navn.size());
  vector<string> navnSmaa(navn.size());
  transform(navn.begin(), navn.end(),
            navnStore.begin(), tilStore);
  transform(navn.begin(), navn.end(),
            navnSmaa.begin(), tilSmaa);

  cout << "\nNavn: ";
  for (i = 0; i < navn.size(); i++) cout << navn[i] << " ... ";
  cout << "\nStore: ";
  for (i = 0; i < navnStore.size(); i++) {
    cout << navnStore[i] << " ... ";
    }
  cout << "\nSmaa: ";
  for (i = 0; i < navnSmaa.size(); i++) {
    cout << navnSmaa[i] << " ... ";
    }

  // Skal nå sortere navnStore. Tar vare på originalen slik
  // at oppgave d) kan løses
  vector<string>navnStoreOrig(navnStore);
  cout << "\n\nOppgave c:\n";
```

```
  sort(navnStore.begin(), navnStore.end(), mindreEnnString);
  cout << "Store, sortert: ";
  for (i = 0; i < navnStore.size(); i++) {
    cout << navnStore[i] << " ... ";
  }

  cout << "\n\nOppgave d:\n";
  if (equal(navn.begin(), navn.end(),
      navnStoreOrig.begin(), likhet)) cout << "1: Likhet!\n";
  if (equal(navn.begin(), navn.end(),
      navnSmaa.begin(), likhet)) cout << "2: Likhet!\n";
  if (equal(navnSmaa.begin(), navnSmaa.end(),
      navnStoreOrig.begin(), likhet)) cout << "3: Likhet!\n";

  return 0;
} // main

// Funksjoner som trengs

// Egne utgaver av tolower() og toupper() (oppgave a))

char tilStor(char tegn)
{
  if (tegn >= 'a' && tegn <= 'z') return toupper(tegn);
  else if (tegn == 'æ') return 'Æ';
  else if (tegn == 'ø') return 'Ø';
  else if (tegn == 'å') return 'Å';
  else return tegn;
} // tilStor

char tilLiten(char tegn)
{
  if (tegn >= 'A' && tegn <= 'Z') return tolower(tegn);
  else if (tegn == 'Æ') return 'æ';
  else if (tegn == 'Ø') return 'ø';
  else if (tegn == 'Å') return 'å';
  else return tegn;
} // tilLit

string tilStore(string tekst)
{
  for (int i = 0; i < tekst.size(); i++) tekst[i] =
tilStor(tekst[i]);
  return tekst;
} // tilStore

string tilSmaa(string tekst)
{
  for (int i = 0; i < tekst.size(); i++) tekst[i] =
tilLiten(tekst[i]);
  return tekst;
} // tilSmaa


// Oppgave b)
bool mindreEnnChar(char c1, char c2)
{
```

```
  const string alfabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ";
  int pos1 = alfabet.find(c1);
  int pos2 = alfabet.find(c2);
  return (pos1 < pos2);
} // mindreEnn

bool mindreEnnString(const string &s1, const string &s2)
{
  return lexicographical_compare(s1.begin(), s1.end(),
s2.begin(), s2.end(), mindreEnnChar);
} // mindreEnnString


// Oppgave c)
bool like(char c1, char c2)
{
  return tilLiten(c1) == tilLiten(c2);
} // like

// Oppgave d)
bool likhet(const string &s1, const string &s2)
{
  return equal(s1.begin(), s1.end(), s2.begin(), like);
} // likhet

/* Utskrift:
Oppgave b:

Navn: Æsop ... Einar ... Åsmund ... Øyvind ...
Store: ÆSOP ... EINAR ... ÅSMUND ... ØYVIND ...
Smaa: æsop ... einar ... åsmund ... øyvind ...

Oppgave c:
Store, sortert: EINAR ... ÆSOP ... ØYVIND ... ÅSMUND ...

Oppgave d:
1: Likhet!
2: Likhet!
3: Likhet!
*/
```

Kapittel 17-6

Oppgave 1

Bytt ut

```
int indeks = finnElevIndeks(navn);
```

med

```
int indeks = find(elever, elever + antElever, navn) - elever;
```

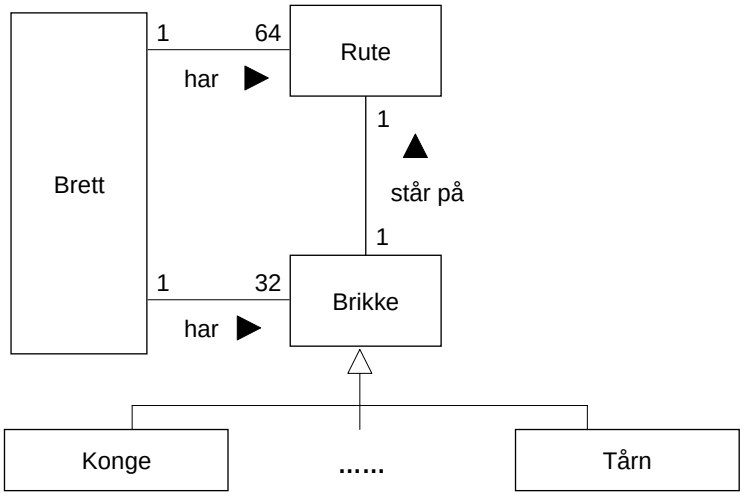
i begge funksjonene.

Oppgave 2

```
bool Elevregister::settFravaer(
  const string &navn, int nyttFravaer)
{
  Elev elev(navn);
  Elev* eleven = find(elever, elever + antElever, elev);
  if (eleven < elever + antElever) {
    eleven->settFravaer(nyttFravaer);
    return true;
    }
  else return false;
} // settFravaer


bool Elevregister::endreNavn(const string &gmlNavn,
                             const string &nyttNavn)
{
  Elev elev(gmlNavn);
  Elev* eleven = find(elever, elever + antElever, elev);
  if (eleven < elever + antElever) {
    eleven->settNavn(nyttNavn);
    return true;
    }
  else return false;
} // endreNavn

bool Elevregister::slettElev(const string &navn)
{
  Elev elev(navn);
  Elev *elevpeker = find(elever, elever + antElever, elev);
  int indeks = elevpeker - elever;
  // Sletter ved å forskyve pekerne en posisjon til venstre
  if (indeks >= 0 && indeks < antElever) {
    for (int i = indeks; i < antElever - 1; i++) {
      elever[i] = elever[i + 1];
      }
    antElever--;
    return true;
    }
  else return false;
} // slettElev
```

Kapittel 18-1

Oppgave 1



Kapittel 18-3

Oppgave 1

Følgende setninger er lovlige: b), e), f), g), i), j) og k).

Oppgave 2

Nå er b), e), f), g) og j) lovlige setninger.

Kapittel 18-4

Oppgave 1

Følgende setninger er lovlige uavhengig av om basisklassen er offentlig eller privat: b), d), e), f), g) og j). Setning k) er bare lovlig dersom basisklassen er offentlig.

Oppgave 2

```
Barometer *baromPeker = &vindmaaler;
```

er ulovlig. De to andre initieringene er lovlige.

Oppgave 3

En pekertabell gjør det mulig å håndtere måledata av forskjellig slag under ett. En tabell kan bare inneholde elementer av samme type. Her er elementtypen **Maaler***. Men siden en peker til basisklassen kan peke til et objekt av en avledet klasse, kan pekerne settes til å peke til forskjellige typer objekter. Ved å gjennomløpe pekertabellen oppnår vi å behandle objekter av forskjellige klasser under ett.

Oppgave 4

Maaler er en abstrakt klasse på grunn av at den inneholder en ekte virtuell funksjon. Det er ikke mulig å lage objekter av abstrakte klasser. Hvis vi bytter ut

```
virtual void skrivData(void) const = 0;
```

med

```
virtual void skrivData(void) const;
```

i klassedefinisjonen, og definerer funksjonen **skrivData( )**, kan vi lage objekter av klassen. Definisjonen legger vi vanligvis i implementasjonsfilen til klassen.

Klassen **Maaler** inneholder ingen måledata, bare nummer og posisjon. Hvis vi antar at brukeren klassifiserer dataene etter type, bør det ikke

være mulig å lage objekter av klassen **Maaler**. Dersom det derimot er ønskelig å registrere målere uten data, bør brukeren ha muligheten til å lage objekter av klassen **Maaler**. Det er derfor den virkeligheten man skal modellere som bestemmer hvorvidt en klasse bør være abstrakt eller ikke.

Oppgave 5

Alle arvede medlemsfunksjoner kan omdefineres i en avledet klasse. For objekter av den avledete klassen bør den utgaven av funksjonen som tilhører den avledete klassen brukes (og ikke den utgaven som arves). Dette gjelder uansett om funksjonen kalles via et objekt av klassen, eller via en peker eller en referanse. Pekeren (referansen) kan være deklarert til å peke (referere) til objekter av basisklassen. Funksjonen må være virtuell dersom vi i de siste tilfellene skal få utført riktig utgave av funksjonen.

Oppgave 6

```
barometer.skrivData(); // tidlig binding, Barometer sin skrivData()
maalerPeker1->skrivData();//sein binding, Barometer sin skrivData()
maalerPeker1->Barometer::skrivData();// kompileringsfeil
maalerPeker2->skrivData(); // sein binding, Maaler sin skrivData()
maalerRef.skrivData(); // sein binding, Barometer sin skrivData()
```

Kapittel 18-5

Oppgave 1

```
Basis::test1
Basis::test2
Basis::test1, tall = 4
Basis::test2, tall = 5
Avledet::test1, tall = 3
Basis::test2
Basis::test2, tall = 7
```

Kapittel 18-6

Oppgave a), b) og c) - reviderte klassedefinisjoner:

```
#ifndef MAALER_DEFINERT
  #define MAALER_DEFINERT
  class Maaler {
    public:
```

```
    void         settNr(
                   int nyttNr);              // Inn
    void         settPosisjon(
                   int nyPosisjon);          // Inn
    int          finnNr() const;
    int          finnPosisjon() const;
    virtual void skrivData() const = 0;

  protected:
    // Oppgave a)
    Maaler(
      int initNr,         // Inn
      int initPosisjon);  // Inn

    // Oppgave c)
    Maaler(
      const Maaler &original); // Inn
    virtual ~Maaler();
    Maaler &operator=(
      const Maaler &original);  // Inn

    void skrivFelles() const;

  private:
    int nr;
    int posisjon;
  }; // Maaler

//-------------------------------------------------------------
class Termometer: public Maaler {
  public:
    void         settTemperatur(
                   double nyTemperatur);  // Inn, Celsius
    double       finnTemperatur() const;
    virtual void skrivData() const;

  protected:
    // Oppgave a)
    Termometer(
      int    initNr,              // Inn
      int    initPosisjon,        // Inn
      double initTemp);           // Inn

    // Oppgave c)
    Termometer(
      const Termometer &original); // Inn
    virtual ~Termometer();
    Termometer &operator=(
      const Termometer &original);  // Inn

  private:
    double temperatur;
  }; // Termometer


//-------------------------------------------------------------
```

```
  // Oppgave b)
  class VaeskeTermometer: public Termometer {
    public:
      VaeskeTermometer(
        int    initNr,                    // Inn
        int    initPosisjon,              // Inn
        double initTemp,                  // Inn
        double initFrysepkt);             // Inn

      // Oppgave c)
      VaeskeTermometer(
        const VaeskeTermometer &original); // Inn
      virtual ~VaeskeTermometer();
      VaeskeTermometer &operator=(
        const VaeskeTermometer &original);  // Inn

      void         settFrysepkt(
                     double nyttFrysepkt);  // Inn
      double       finnFrysepkt() const;
      virtual void skrivData() const;

    private:
      double frysepunkt;
    }; // VaeskeTermometer

#endif
```

Oppgave a) - implementasjon av nye medlemsfunksjoner

```
Maaler::Maaler(int initNr, int initPosisjon)
      : nr(initNr),
        posisjon(initPosisjon)
{ } // Maaler
//------------------------------------------------------------

Termometer::Termometer(int initNr, int initPosisjon,
                       double initTemp)
          : Maaler(initNr, initPosisjon),
            temperatur(initTemp)
{ } // Termometer
```

Oppgave b) - implementasjon av nye medlemsfunksjoner

```
//------------------------------------------------------------
VaeskeTermometer::VaeskeTermometer(int initNr,
                    int initPosisjon,
                    double initTemp, double initFrysepkt)
                : Termometer(initNr, initPosisjon, initTemp),
                  frysepunkt(initFrysepkt)
{ } // VaeskeTermometer
//------------------------------------------------------------
void VaeskeTermometer::settFrysepkt(double nyttFrysepkt)
{
  frysepunkt = nyttFrysepkt;
} // settFrysepkt
```

```
//-------------------------------------------------------------
double VaeskeTermometer::finnFrysepkt() const
{
  return frysepunkt;
} // finnFrysepkt

//-------------------------------------------------------------
void VaeskeTermometer::skrivData() const
{
  Termometer::skrivFelles();
  cout <<
    "Nærmere bestemt et væsketermometer med frysepunkt " <<
    frysepunkt << " gr. C.\n";
} // skrivData
```

Oppgave c) - implementasjon av nye medlemsfunksjoner

```
//-------------------------------------------------------------
Maaler::Maaler(const Maaler &original)
{
  cout << "Klassen Maaler, kopikonstruktør\n";
  nr = original.nr;
  posisjon = original.posisjon;
} // Maaler

//-------------------------------------------------------------
Maaler::~Maaler()
{
  cout << "Klassen Maaler, destruktør\n";
} // ~Maaler

//-------------------------------------------------------------
Maaler &Maaler::operator=(const Maaler &original)
{
  cout << "Klassen Maaler, tilordningsoperator.";
  if (&original != this) {
    cout << " .. &original != this .. \n";
    nr = original.nr;
    posisjon = original.posisjon;
  }
  return *this;
} // operator=

//-------------------------------------------------------------
Termometer::Termometer(const Termometer &original)
          : Maaler(original)
{
  cout << "Klassen Termometer, kopikonstruktør\n";
  temperatur = original.temperatur;
} // Termometer

//-------------------------------------------------------------
Termometer::~Termometer()
{
  cout << "Klassen Termometer, destruktør\n";
} // ~Termometer
```

```
//-------------------------------------------------------------
Termometer &Termometer::operator=(const Termometer &original)
{
  cout << "Klassen Termometer, tilordningsoperator.";
  if (&original != this) {
    cout << " .. &original != this .. \n";
    Maaler::operator=(original);
    temperatur = original.temperatur;
    }
  return *this;
} // operator=

//-------------------------------------------------------------
VaeskeTermometer::VaeskeTermometer(
                      const VaeskeTermometer &original)
               : Termometer(original)
{
  cout << "Klassen VaeskeTermometer, kopikonstruktør\n";
  frysepunkt = original.frysepunkt;
} // VaeskeTermometer

//-------------------------------------------------------------
VaeskeTermometer::~VaeskeTermometer()
{
  cout << "Klassen VaeskeTermometer, destruktør\n";
} // ~VaeskeTermometer

//-------------------------------------------------------------
VaeskeTermometer &VaeskeTermometer::
                 operator=(const VaeskeTermometer &original)
{
  cout << "Klassen VaeskeTermometer, tilordningsoperator.";
  if (&original != this) {
    cout << " .. &original != this .. \n";
    Termometer::operator=(original);
    frysepunkt = original.frysepunkt;
    }
  return *this;
} // operator=
```

Oppgave d)

```
#include <iostream>
using namespace std;

#include "Maaler.h"

int main()
{
  VaeskeTermometer t1(11, 100, 15.6, -38);
  VaeskeTermometer t2(12, 101, 17.6, -42);

  cout << "T1: ";
  t1.skrivData();
  cout << "T2: ";
  t2.skrivData();
```

```
  // Kopikonstruktøren
  VaeskeTermometer t3 = t1;
  cout << "T3 (er lik T1): ";
  t3.skrivData();

  // Tilordningsoperatoren
  t3 = t2;
  cout << "T3 (er lik T2): ";
  t2.skrivData();

  return 0;
} // main

/* Utskrift - Kaller ikke tilordningsoperatoren i basisklassen
T1: MÅler nr. 11 er plassert pÅ posisjon nr. 100
Nµrmere bestemt et vµsketermometer med frysepunkt -38 gr. C.
T2: MÅler nr. 12 er plassert pÅ posisjon nr. 101
Nµrmere bestemt et vµsketermometer med frysepunkt -42 gr. C.
Klassen Maaler, kopikonstruktør
Klassen Termometer, kopikonstruktør
Klassen VaeskeTermometer, kopikonstruktør
T3 (er lik T1): MÅler nr. 11 er plassert pÅ posisjon nr. 100
Nµrmere bestemt et vµsketermometer med frysepunkt -38 gr. C.
Klassen VaeskeTermometer, tilordningsoperator. .. &original !=
this ..
T3 (er lik T2): MÅler nr. 12 er plassert pÅ posisjon nr. 101
Nµrmere bestemt et vµsketermometer med frysepunkt -42 gr. C.
Klassen VaeskeTermometer, destruktør
Klassen Termometer, destruktør
Klassen Maaler, destruktør
Klassen VaeskeTermometer, destruktør
Klassen Termometer, destruktør
Klassen Maaler, destruktør
Klassen VaeskeTermometer, destruktør
Klassen Termometer, destruktør
Klassen Maaler, destruktør

- Kaller tilordningsoperatoren i basisklassen
T1: MÅler nr. 11 er plassert pÅ posisjon nr. 100
Nµrmere bestemt et vµsketermometer med frysepunkt -38 gr. C.
T2: MÅler nr. 12 er plassert pÅ posisjon nr. 101
Nµrmere bestemt et vµsketermometer med frysepunkt -42 gr. C.
Klassen Maaler, kopikonstruktør
Klassen Termometer, kopikonstruktør
Klassen VaeskeTermometer, kopikonstruktør
T3 (er lik T1): MÅler nr. 11 er plassert pÅ posisjon nr. 100
Nµrmere bestemt et vµsketermometer med frysepunkt -38 gr. C.
Klassen VaeskeTermometer, tilordningsoperator. .. &original !=
this ..
Klassen Termometer, tilordningsoperator. .. &original != this
..
Klassen Maaler, tilordningsoperator. .. &original != this ..
T3 (er lik T2): MÅler nr. 12 er plassert pÅ posisjon nr. 101
Nµrmere bestemt et vµsketermometer med frysepunkt -42 gr. C.
Klassen VaeskeTermometer, destruktør
Klassen Termometer, destruktør
Klassen Maaler, destruktør
```

```
Klassen VaeskeTermometer, destruktør
Klassen Termometer, destruktør
Klassen Maaler, destruktør
Klassen VaeskeTermometer, destruktør
Klassen Termometer, destruktør
Klassen Maaler, destruktør
*/
```

Kapittel 18-7

Funksjonen kan se slik ut:

```
bool smlknMaalere(const Maaler *m1, const Maaler *m2) {
  return m1->finnNr() < m2->finnNr();
}
```

Vi har en vektor med pekere til måler-objekter:

```
vector<Maaler *> maalere;
```

Nå kan vi sortere disse slik:

```
sort(maalere.begin(), maalere.end(), smlknMaalere);
```

Kapittel 18-8

Oppgave a)

Skriver ut bokstaven i stedet for stjerne, det vil si at den store A'en skrives ut med A'er, den store B'en med B'er, osv.

```
#include <iostream>
using namespace std;

class Bokstav {

  public:
    void         settTegn(
                   char nyttTegn);           // Inn
    char         finnTegn(void) const;
    virtual void tegnBokstav(void) const = 0;

  protected:
      Bokstav(char initTegn);

  private:
    char tegn;
  }; // Bokstav
```

```
//-------------------------------------------------------------
class BokstavA: public Bokstav {
  public:
    BokstavA(char initTegn);
    virtual void tegnBokstav(void) const;     // Inn
  }; // BokstavA

//-------------------------------------------------------------
class BokstavN: public Bokstav {
  public:
    BokstavN(char initTegn);
    virtual void tegnBokstav(void) const;     // Inn
  }; // BokstavN

//-------------------------------------------------------------
Bokstav::Bokstav(char initTegn)
       : tegn(initTegn)
{ } // Bokstav

//-------------------------------------------------------------
void Bokstav::settTegn(char nyttTegn)
{
  tegn = nyttTegn;
} // settTegn



//-------------------------------------------------------------
char Bokstav::finnTegn(void) const
{
  return tegn;
} // finnTegn

//-------------------------------------------------------------
BokstavA::BokstavA(char initTegn)
       : Bokstav(initTegn)
{ } // BokstavA

//-------------------------------------------------------------
BokstavN::BokstavN(char initTegn)
       : Bokstav(initTegn)
{ } // BokstavN

//-------------------------------------------------------------
void BokstavA::tegnBokstav(void) const
{
  char tegnet = finnTegn();
  cout << tegnet << tegnet << tegnet <<
          tegnet << tegnet << endl;
  cout << tegnet << "   " << tegnet << endl;
  cout << tegnet << tegnet << tegnet <<
          tegnet << tegnet << endl;
  cout << tegnet << "   " << tegnet << endl;
  cout << tegnet << "   " << tegnet << endl;
} // tegnBokstav
```

```
//------------------------------------------------------------
void BokstavN::tegnBokstav(void) const
{
  char tegnet = finnTegn();
  cout << tegnet << "   " << tegnet << endl;
  cout << tegnet << tegnet << "  " << tegnet << endl;
  cout << tegnet << ' ' << tegnet << ' ' << tegnet << endl;
  cout << tegnet << "  " << tegnet << tegnet << endl;
  cout << tegnet << "  " << tegnet << tegnet << endl;
  cout << tegnet << "   " << tegnet << endl;
} // tegnBokstav

//------------------------------------------------------------
```

Oppgave b)

```
void main()
{
  BokstavA storA('A');
  storA.tegnBokstav();
  cout << endl << endl;
  BokstavN storN('N');
  storN.tegnBokstav();
  cout << endl << endl;
  storN.tegnBokstav();
  cout << endl << endl;
  storA.tegnBokstav();
} // main
```

Oppgave c)

I klassedefinisjonen:

```
virtual void tegnBokstav(void) const;
```

Implementasjonen:

```
void Bokstav::tegnBokstav(void) const {
  cout << "Ikke implementert\n";
}
```