

Norges teknisk-naturvitenskapelige
universitet
Institutt for datateknologi og
informatikk

TDT4102 Prosedyre-
og objektorientert
programmering
Vår 2024

Øving 2

Innleveringsfrist: 2024-01-26

Mål for denne øvingen:

- Lære hvordan programmer kan lese inn data fra brukeren og skrive tekst ut på skjermen
- Lære hvordan funksjoner kan ta inn data som argumenter og returnere verdier
- Lære grunnleggende grafikk
- Lære om løkker
- Lære om `vector`

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Teorioppgaver besvares med kommentarer i kildekoden slik at læringsassistenten enkelt finner svaret ved godkjenning.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.
- Benytt VSCode til å skrive, kompilere og kjøre kode.

Anbefalt lesestoff:

- Kapittel 2, 3 og 4 i læreboken.
- Dokumentasjon til grafikkbiblioteket `AnimationWindow`. Dette kan du finne [her](#).

1 Menysystem, Funksjoner og «Input/Output» (25%)

I koden under er det definert tre funksjoner. `main()` må være med i alle programmer, den blir kalt av operativsystemet som starter kjøringen av programmet. Videre kan du lage så mange funksjoner du måtte ønske, `add()` og `inputIntegersAndPrintProduct()` er eksempler på brukerdefinerte funksjoner.

```
#include "std_lib_facilities.h"

// Funksjonen heter 'add', den tar inn to heltall, legger sammen
// tallene og returnerer resultatet, som er et heltall
int add(int a, int b) {
    return a + b;
}

// Funksjonen tar ikke inn noe og returnerer ingenting
// Input kommer fra brukeren av programmet: cin,
// og output skrives til brukeren av programmet: cout.
void inputIntegersAndPrintProduct() {
    int x = 0;
    int y = 0;

    cout << "Skriv inn to heltall: ";
    cin >> x;
    cin >> y;

    int product = x * y;
    cout << x << " * " << y << " = " << product << endl;
}

int main() {
    int sumOfOneAndTwo = add(1, 2);
    cout << "1 + 2 = " << sumOfOneAndTwo << endl;
    cout << "2 + 2 = " << add(2, 2) << endl;
    inputIntegersAndPrintProduct();
    return 0;
}
```

Opprett et prosjekt som beskrevet i øving 0.

- a) I øving 1 har vi testet funksjonene vi har skrevet på en ganske usystematisk måte, ved å prøve dem ut i `main`. Dette skal vi nå rydde opp i. **Lag et menysystem i `main` der brukeren kan velge mellom funksjoner den ønsker å kjøre.** Legg inn funksjonene du lager utover i øvingen og test dem gjennom menyen. Etter noen oppgaver kan den for eksempel se slik ut:

Velg funksjon:

0) Avslutt

1) Summer to tall

2) Summer flere tall

3) Konverter NOK til EURO.

Angi valg (0-3):

Hvis brukeren f.eks. velger 2, skal funksjonen som lar brukeren angi tall for summering bli utført, når denne er ferdig, skal menyen vises på nytt. Programmet skal ikke avslutte før brukeren selv velger dette ved å angi menyvalget for «Avslutt».

Tips: Sjekk ut "Switch case" i infobanken til TDT4102 i VS-Code.

- b) **Definer en funksjon `inputAndPrintInteger`** som lar brukeren skrive inn et heltall og skriver det ut til skjerm. Test funksjonen fra `main`, ved å legge det til i testmenyen.

Eksempel på resultat av å kjøre programmet:

```
>> inputAndPrintInteger();
```

Skriv inn et tall: 42

Du skrev: 42

- c) **Definer en funksjon `inputInteger`** som lar brukeren skrive inn et heltall og *returnerer* dette. Test funksjonen fra `main`, ved hjelp av testmenyen.

Eksempel på resultat av å kjøre programmet:

```
>> int number = inputInteger();
```

```
>> cout << "Du skrev: " << number;
```

Skriv inn et tall: 123

Du skrev: 123

- d) **Definer en funksjon `inputIntegersAndPrintSum`** som ved å bruke en av funksjonene du nå har skrevet, leser inn to heltall fra brukeren og skriver ut kun summen til skjermen. Husk å legge til funksjonen i testmenyen.

Eksempel på resultat av å kjøre programmet:

Skriv inn et tall: 3

Skriv inn et tall: 4

Summen av tallene: 7

- e) **Teori:** hvorfor valgte du å benytte `inputAndPrintInteger` eller `inputInteger` framfor den andre i deloppgave d)?

- f) **Definer en funksjon `isOdd`**, som tar inn et heltall som parameter og returnerer en boolsk verdi. Funksjonen skal returnere `true` dersom argumentet er et oddetall og `false` ellers. Den skal *ikke* lese noe inn fra brukeren eller skrive ut noe til skjermen. Test funksjonen på alle heltall i intervallet $[0, 15]$ i testmenyen.

Tips:

C++ tolker `true` og `false` henholdsvis som 1 og 0. Hvis du ønsker å skrive "`true`" og "`false`" til skjermen, kan du kalle funksjonen med `cout << (isOdd(x) ? "true" : "false");`

2 Løkker (15%)

- a) **Definer en funksjon**, som skal lese inn et bestemt antall heltall fra brukeren, og skrive summen av disse heltallene til skjermen. Antall tall som skal summeres skal brukeren angi i starten av programmet.
- b) **Definer en funksjon**, som skal lese inn heltall fra brukeren, helt til brukeren skriver inn tallet 0. Summen av heltallene skal skrives til skjermen.
- c) **Teori:** De to forrige deloppgavene skal løses med løkker. Gjør kort rede for hvilken type løkke som er best egnet for hver av deloppgavene.
- d) **Definer funksjonen `inputDouble`** som skal gjøre det samme som `inputInteger` fra oppgave 1, men istedenfor å lese inn et heltall, skal denne funksjonen lese inn og returnere et desimaltall.
- e) **Definer en funksjon som konverterer fra Norske kroner NOK til Euro.**
 La brukeren gi beløpet som skal konverteres som et positivt desimaltall. Vekslingskurs kan du bestemme selv, f.eks. 9.75. Hvis brukeren gir et negativt tall skal programmet spørre etter et nytt tall. Skriv ut det vekslede beløpet med to desimaler.
 Gjenbruk funksjoner så langt som mulig, og test funksjonen fra `main()`.
Hint: For å spesifisere at reelle tall (flyttall) skal skrives ut med et bestemt antall desimaler kan du benytte `setprecision` og `fixed`. Bokens Appendix har mye nyttig informasjon: her er B.7.6 til god hjelp. Kapittel 11.2.3 og 11.2.4 har flere eksempler.
- f) **Teori:** forklar hvorfor du bør bruke `inputDouble` framfor `inputInteger` i deloppgave e). Kommenter også valg av returtypen til funksjonen.
- g) **Definer en funksjon som skriver ut en gangetabell** på skjermen (`cout`). La brukeren gi både bredde og høyde på tabellen. Velg selv navn for funksjonen. *Slå opp i boken på `setw` for å finne såkalte manipulatorer som hjelper til med å skrive pene tabeller.* Legg denne funksjonen til i testmenyen.

Eksempel på resultat av å kjøre programmet:

Enter height: 3

Enter width: 4

1	2	3	4
2	4	6	8
3	6	9	12

3 Bruk av funksjoner i funksjoner, og røtter (20%)

I funksjonene under skal du selv avgjøre hva som skal være parameter(e).

- a) Definer en funksjon **discriminant** som regner ut

$$b^2 - 4ac$$

og returnerer verdien (ingen utskrift til skjerm). a , b og c er alle av typen `double`.

Tips:

For å regne ut x^y i C++ kan man bruke `pow(x,y)`

- b) Definer en funksjon **printRealRoots** som finner de reelle røttene til andregradsligningen

$$ax^2 + bx + c = 0$$

ved å bruke formelen

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

der a , b og c er gitt som argumenter til funksjonen. Gjenbruk funksjonene fra de forrige deloppgavene, og skriv ut løsningene til skjerm.

Hint: Bruk en funksjon fra standardbiblioteket til å finne kvadratroten. Bokens Appendix B er nyttig å slå opp i for å finne den informasjonen. B.9.2 Standard mathematical functions er særlig nyttig i denne oppgaven.

Ligningen har 2, 1, eller 0 reelle løsninger (vi ser bort fra imaginære løsninger). Dette bestemmes ved at:

2 løsninger dersom	$b^2 - 4ac > 0$
1 løsning dersom	$b^2 - 4ac = 0$
ingen løsninger dersom	$b^2 - 4ac < 0$

Du kan anta at $a \neq 0$.

- c) Lag en funksjon **solveQuadraticEquation** som lar brukeren skrive inn 3 desimaltall og bruk **printRealRoots** til å regne ut røttene til andregradsuttrykket gitt ved disse tallene.
- d) Legg til **solveQuadraticEquation** i testmenyen i **main()**.
- e) Bruk testmenyen til å finne røttene til de tre andregradsligningene gitt nedenfor. Sjekk at programmet fungerer med tall som gir 0, 1, og 2 løsninger. Ligningene under gir henholdsvis 0, 1 og 2 reelle løsninger.

$$x^2 + 2x + 4 = 0$$

$$4x^2 + 4x + 1 = 0$$

$$8x^2 + 4x - 1 = 0$$

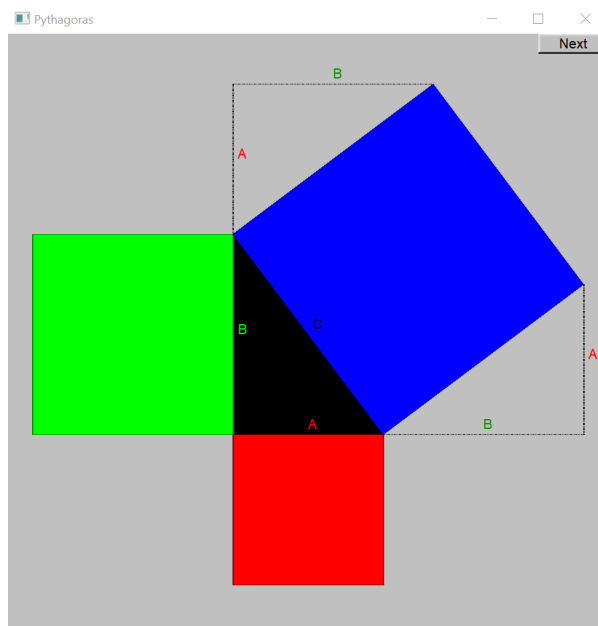
Generelt i øvingsopplegget bør du gjøre noe tilsvarende for å teste at koden din fungerer som den skal.

4 Grafikk - Pythagoras' læresetning (25%)

I denne oppgaven skal du tegne en visualisering av Pythagoras' læresetning. Pythagoras' læresetning sier at i en rettvinklet trekant er summen av kvadratet av hver katet lik kvadratet av hypotenusen. Den kan også uttrykkes slik:

$$a^2 + b^2 = c^2$$

Her er a og b katetene og c hypotenusen.



Figuren over viser et forslag til løsning av oppgaven. De stiplede linjene er ikke en del av oppgaven. De er der kun som en visuell hjelp. Det kan også være lurt å sjekke ut dokumentasjonen til grafikkbiblioteket vi benytter, som ligger [her](#).

a) Lag funksjonen pythagoras. I denne funksjonen skal all tegningen foregå.

For å gjøre dette må du inkludere `AnimationWindow.h`. Dette gjør du ved å skrive `#include "AnimationWindow.h"` øverst i main-filen.

b) Opprette et vindu

Det første man må gjøre når man skal tegne på skjermen er å opprette et vindu. Dette gjør du ved å opprette en variabel av typen `AnimationWindow`. For å lage en `AnimationWindow` variabel med navn `win` skriver du:

```
AnimationWindow win{100,100, w, h, "Pythagoras"};
```

De første argumentene 100,100, er hvor hjørnet oppe til venstre av vinduet blir plassert på skjermen. `w` er bredden på vinduet og `h` er høyden. Disse bestemmer du selv, ved å bytte ut `w` og `h` med heltall. Det siste argumentet er tittelen til vinduet.

For at vinduet skal holde seg åpent må du skrive: `win.wait_for_close();` som siste linje av funksjonen. Da vil vinduet holde seg åpent frem til brukeren lukker det.

c) Tegning av en rettvinklet trekant. Du skal først tegne trekanten. Opprett variabelene `point1`, `point2`, og `point3` med typen `Point`. Hver av disse er et hjørne til vår trekant. Ved å bruke `Point point1{x, y};` kan du definere hvor posisjonen til hjørnene skal være. Husk at (0,0) er øverst til venstre i vinduet.

Siden det senere skal tegnes kvadrater ut fra hver side av trekanten må du passe på at det er plass til disse i vinduet.

Å tegne trekanten gjør du ved å skrive:

```
win.draw_triangle(point1, point2, point3, Color::red);
```

Du gir trekanten en annen farge ved å endre `red` i `Color::red` til den fargen du har lyst på.

d) Tegning av kvadrater. Kvadratene skal tegnes med:

```
win.draw_quad(Point{x, y}, Point{x, y}, Point{x, y}, Point{x, y}, Color::red);
```

For hver side i trekanten skal det tegnes et kvadrat som deler en side med siden i trekanten.

Det går fint om firkantene ikke er helt perfekte, dette er kun ment som en innføringsoppgave i grafikk.

Når vi tegner figurer er det viktig at alle punktene enten tegnes med eller mot klokka. Ingen av linjene i figuren kan krysse hverandre, og alle punktene kan heller ikke ligge på en linje. Hvis de gjør dette vil programmet ditt krasje.

5 Renter med bruk av løkker (15%)

I denne oppgaven skal vi se på renter og rentes renter når man gjør et bankinnskudd. Formlene er oppgitt med antagelsen om at renter er oppgitt som heltall, så 30 betyr 30% rente.

Nyttig å vite: kort om vector

```
vector<int> t(10); // 10 int av verdien 0
for (int i = 0; i < 10; i++) {
    t.at(i) = i + 93;
    cout << t.at(i) << endl;
}
```

For å kunne transformere data er det nyttig å samle data i beholdere. Det finnes mange måter å lagre data på, og den mest anvendelige beholderen i C++ er `vector`. En `vector` kan holde en samling data av en gitt type. `vector<int>` utretter følgende: det opprettes en `vector`, den heter `intVector` og kan holde på verdier av typen `int`. På samme måte som at `vector<double>` `doubleVector` oppretter en `vector` som kan holde `double`-verdier og samlingen heter `doubleVector`. Idet `vector`en opprettes kan også et antall elementer av typen initialiseres til standardverdien for typen, slik det er gjort i eksemplet ovenfor. For de innebygde typene er det som regel 0, 0.0, osv. For å legge til et element i `vector`en kan `push_back` benyttes på samlingen.

F.eks. `intVector.push_back(32)` legger til elementet 32 i samlingen.

For å finne ut hvor mange elementer det er i samlingen kan `.size()` benyttes på samlingen.

Nyttig å vite: heltall, flyttall og `static_cast`

Når man utfører divisjon der begge operandene er heltall vil resultatet være et heltall. Der resultatet av divisjonen er avrundet mot null. F.eks. $9/10 = 0$, $-8/5 = -1$.

Minst en av operandene må være et flyttall hvis man vil ha et flyttall som resultat av en divisjon. Hvis man vil ha et flyttalls resultat av en divisjon der begge operandene er heltall kan man *caste* en av operandene om til et flyttall med `static_cast<double>()`. F.eks. `static_cast<double>(9)/10` blir 0.9 og `-8/static_cast<double>(5)` blir -1.6.

- a) **Skriv funksjonen `calculateBalance`**, denne skal regne ut den årlige saldoen på konto over et gitt antall år. Den skal ta inn innskuddet, renten (heltall) og antall år. Funksjonen skal returnere en heltallsvektor som angir saldoen for hvert år. Regn det første elementet i vektoren som år 0, og la dette være innskuddet, anta at kontoen har 0 kr i saldo før innskuddet.

Det er viktig å huske på rentes rente, økningen i saldo blir ikke lik hvert år, fordi vi vil få renter på rentene når de blir satt inn etter hvert år. Du trenger kun å regne med at rentene blir satt inn en gang på slutten av hvert år. Saldoen hvert år kan dermed beregnes med følgende formel:

$$\text{Saldo} = \text{Innskudd} * (1 + \text{Rente}/100)^{\text{AntallÅr}}$$

Eksempel: Et innskudd på 5 000 kroner og en rente på 3% vil etter første år gi en saldo på

$$5000 * (1 + 3/100)^1 = 5150 \text{ kroner}$$

Etter to år vil saldoen være

$$5000 * (1 + 3/100)^2 = 5304,5 \text{ kroner}$$

Merk: selv om det returneres en heltallsvektor krever noen av kalkulasjonene at vi bruker flyttall, les mer om hvordan du løser dette i «Nyttig å vite: heltall, flyttall og `static_cast`».

- b) **Skriv funksjonen `printBalance`**. Funksjonen skal ta inn en heltallsvektor med bankkontosaldoer. Den skal skrive disse ut på en pen og leselig måte til skjermen, slik at brukeren enkelt kan se saldoens utvikling hvert år dersom pengene får stå uberørt. Utskriften skal formatteres som vist under:

År	Saldo
0	5000
1	5150
...	
x	x

Denne tabellen skal kunne skrives ut fra testmenyen.

Slå opp i B.7.6 i boken for nyttige metoder å formattere utskrift på, se spesielt på funksjonen `setw` som også står beskrevet i Kapittel 11.2.5.

- c) **Oppgave om feilmeldinger**

En vanlig feil som gjøres når man prøver å skrive ut innholdet i en `vector` gir følgende feilmelding ved kompilering:

```
no operator "<<" matches these operands -- operand types are:
std::ostream << std::vector<int, std::allocator<int>>
```

Gi en mulig forklaring på hva feilen skyldes.

- d) **Debuggeroppgave**

Følgende kode foreslås for å skrive ut innholdet i en `vector<int>`.

```
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9};
for (int i = 0; i <= v.size(); i++) {
    cout << v.at(i) << endl;
}
```

Kjør kodesnutten i `main()` og les feilmeldingen som dukker opp under kjøring. Hvor i koden ligger feilen, og hva forårsaker den? **Tips:** Du kan bruke debuggerne for å se hva som skjer i hver iterasjon, og dermed letter finne ut hvor feilen oppstår og hva den skyldes.