Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2020

Øving 2

Frist: 2020-01-24

Mål for denne øvingen:

- Lære grunnleggende C++ og prosedyreorientert programmering
- 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

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.
- Det anbefales å benytte et IDE (Visual Studio Code).

Anbefalt lesestoff:

- Kapittel 1, 2, 3, 4 og 12 i PPP.

1 Funksjoner og «Input/Output» (20%)

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 << '\n';
}

int main() {
    int sumOfOneAndTwo = add(1, 2);
    cout << "1 + 2 = " << sumOfOneAndTwo << '\n';
    cout << "2 + 2 = " << add(2, 2) << '\n';
    inputIntegersAndPrintProduct();
}
```

Opprett et prosjekt som beskrevet i øving 0.

a) **Definer en funksjon `inputAndPrintInteger`** som lar brukeren skrive inn et heltall og skriver det ut til skjerm. Test funksjonen fra `main`, slik som dette:

```
int main() {
    inputAndPrintInteger();
}
```

Eksempel på resultat av å kjøre programmet:

```
Skriv inn et tall: 42
Du skrev: 42
```

b) **Definer en funksjon `inputInteger`** som lar brukeren skrive inn et heltall og *returnerer* dette. Test funksjonen fra `main`, slik som dette:

```
int main() {
    int number = inputInteger();
    cout << "Du skrev: " << number;
}
```

Eksempel på resultat av å kjøre programmet:

```
Skriv inn et tall: 123
Du skrev: 123
```

c) **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. Eksempel på resultat av å kjøre programmet:

```
Skriv inn et tall: 3
Skriv inn et tall: 4
Summen av tallene: 7
```

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

e) **Definer en funksjon `isOdd`**, som tar inn et heltall 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].

f) **Definer en funksjon `printHumanReadableTime`** som har antall sekunder som parameter, og skriver dette ut til skjerm i et vanlig leselig format. For eksempel vil 10 000 sekunder skrives ut som «2 timer, 46 minutter og 40 sekunder». Programmet ditt kan fravike korrekt grammatikk. «1 minutt» og «1 minutt*er*» er like gode svar i denne oppgaven.

2 Løkker (10%)

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 tallene 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 (flyttal) 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 d). Kommenter også valg av returtypen til funksjonen.

3 Menysystem (10%)

a) Hittil 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. **Skriv om `main` slik at brukeren kan velge i en meny mellom funksjonene fra foregående oppgaver**, eksempel:

```
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».

b) **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.

4 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 alle er av typen `double`.

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.

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

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

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

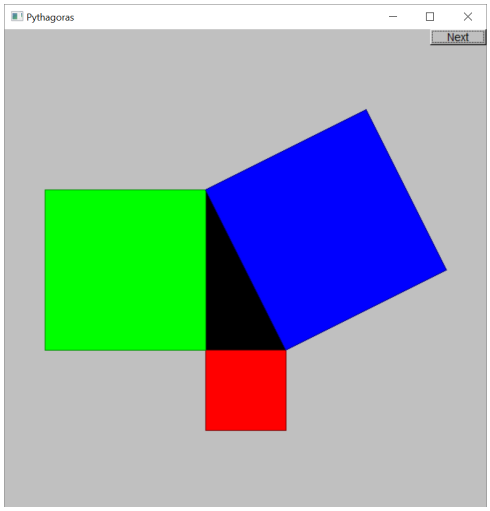
NB: Generelt i øvingsopplegget bør dere gjøre noe tilsvarende for å teste at koden deres fungerer som den skal.

5 Grafikk - Pythagoras' læresetning (20%)

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

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

Her er `a` og `b` katetene og `c` hypotenusen. Kapittel 12.2-12.3 er nyttig for grafikk brukt i denne oppgaven.



a) Lag funksjonen pythagoras. I denne funksjonen skal all tegningen foregå. For at du skal kunne bruke de grafikktypene man trenger i denne oppgaven uten problemer må du skrive `using namespace Graph_lib;` øverst i funksjonen. Du vil lære om `namespace` senere i kurset.

Du må i tillegg inkludere `Graph.h` og `Simple_window.h`. Dette gjør du ved å skrive `#include "Graph.h"` og `#include "Simple_window.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 `Simple_Window`. For å lage en `Simple_Window` variabel med navn `win` skriver du:

```
Simple_window win{Point{100,100},w,h,"Pythagoras"};
```

Det første argumentet `Point{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 opprettes må du skrive: `win.wait_for_button();` som siste linje av funksjonen. Denne linjen gir vinduet kontrollen over programmet til du trykker på next-knappen i vinduet.

c) Tegning av en rettvinklet trekant. Du skal tegne trekanten med typen `Polygon`. Opprett variablen `rightTriangle`. For å legge til hjørner bruker man `.add(Point{x,y})`. Der `Point{x,y}` er prosisjonen til hjørnet. Du skal legge til tre hjørner for å lage trekanten. 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 i vinduet til disse.

Du gir trekanten en farge ved å bruke `.set_fill_color(Color::red)`. Her kan `red` endres til den fargen du har lyst på.

For at trekanten skal tegnes må den festes til vinduet. Dette gjør du ved å skrive: `win.attach(rightTriangle);`

d) Tegning av kvadrater. Kvadratene skal også tegnes med `Polygon`. For hver side i trekanten skal det tegnes et kvadrat som deler en side med siden i trekanten.

6 Lån med bruk av løkker (20%)

I denne oppgaven skal vi se på de to vanligste formene for lån og nedbetaling, nemlig serielån og annuitetslån. Formlene er oppgitt med antagelse om at renter oppgis i heltall, slik at heltallet 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[i] = i + 93;
    cout << t[i] << '\n';
}
```

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> intVector` 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) Serielån. Skriv funksjonen `calculateSeries` som skal regne ut årlige innbetalinger for et lån over et gitt antall år. Funksjonen skal ta inn lånebeløp, rente og antall år, som alle er heltall. Funksjonen skal returnere en heltallsvector med innbetalingsbeløpene. Regn ut innbetalingene ved å benytte formelen:

$$\text{Innbetaling} = \frac{\text{TotaltLån}}{\text{AntallAvdrag}} + \frac{\text{Rente}}{100} * \text{GjenståendeLån}$$

Du trenger kun å regne med en innbetaling i året, og denne skjer ved slutten av året.

Innbetaling er hele utgiften av lånet. Innbetalingen består av avdrag og renter:

- Avdrag: det du betaler ned på lånet. Når avdrag betales blir lånet mindre, du skylder banken mindre penger.
- Renter: prisen på lånet, det beløpet banken tar betalt for at du skal få låne penger. Å betale renter vil ikke påvirke lånets størrelse.

For serielån er avdraget det samme for hver innbetaling. Etter hver innbetaling reduseres lånet med samme beløp som avdraget.

Eksempel: et lån på 10 000 kroner skal nedbetales over en periode på 10 år og renten er 5%.
Etter første år skal renter på 5% * 10 000 kroner = 500 kroner og avdrag på 10 000 / 10 = 1 000 kroner betales til banken. Det betyr at det gjenstår 9 000 kroner av lånet og det du har betalt til banken er 1 500 kroner.
Etter år nummer to skal det kun betales renter av 9 000 kroner som er 450 kroner og nedbetalingen er fortsatt 1 000 kroner. Det blir totalt 1 450 kroner.
osv.

Merk: selv om det returneres en heltallsvector må noen beregninger utføres på flyttall. Resultat av `calculateSeries(10000,5,10)`:

```
År          Innbetaling
1             1500
2             1450
3             1400
4             1350
5             1300
6             1250
7             1200
8             1150
9             1100
10            1050
```

b) Annuitetslån. Definer funksjonen `calculateAnnuity` som regner ut annuitetslån. En formel for dette er:

$$\text{Innbetaling} = \text{TotaltLån} * \frac{\text{Rente}/100}{1 - (1 + \text{Rente}/100)^{-\text{AntallAvdrag}}}$$

Annuitetslån har like store innbetalingsbeløp hver termin. Både avdrag og rentekostnad vil variere for hvert år. Funksjonen skal returnere en heltallsvector på samme måte som i forrige deloppgave, og ta inn det samme.

Eksempel: et lån på 10 000 kroner skal nedbetales over en periode på 10 år og renten er 5%. Det gir innbetalingsbeløp som skal være 1 295 kroner etter å ha satt det inn i funksjonen over. Det betyr at det hvert år skal betales inn 1 295 kroner, uavhengig av hvor mye som er igjen av lånet.
Etter første år vil 5% * 10 000 kroner = 500 kroner være renter og 1 295 - 500 = 795 kroner være avdraget. Det gjenstår nå 10 000 - 795 = 9 205 kroner av lånet.
Etter andre år vil renteandelen være 5% * 9 205 = 460 kroner og avdraget bli 1 295 - 460 = 835 kroner. Det gjenstår nå 9 205 - 835 = 8 370 kroner av lånet.
osv.

Resultat av `calculateAnnuity(10000,5,10)`

```
År         Innbetaling
1            1295
2            1295
3            1295
4            1295
5            1295
6            1295
7            1295
8            1295
9            1295
10           1295
```

c) **Definer en funksjon som skriver ut innbetalingene til hver av planene og sammenligner årlig og totalt beløp.** Dersom vi velger en rente på 3%, et totalt lån på 10 000 kroner og 10 avdrag, så skal de første linjene og den siste linjen i tabellen se slik ut:

```
År        Annuitet          Serie     Differanse
  1           1172           1300           -128
  2           1172           1270            -98
 ...
Total        11720          11650             70
```

Denne tabellen skal kunne skrives ut fra testmenyen.
Slå opp i boken på `setw` *for å finne funksjon(er) som gjør formatering av tabeller lettere.*