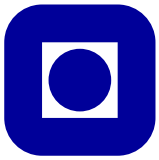

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2019

Øving 2

Frist: 2019-01-25

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

Generelle krav og anbefalinger:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Teorioppgaver besvares med kommentarer i kildekoden slik at læringsassistenten enkelt finner svaret ved godkjenning.
- Det anbefales å benytte et IDE (Visual Studio, XCode).
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.

Anbefalt lesestoff:

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

1 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 << '\n';
}

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

Opprett et prosjekt om inkluderer `std_lib_facilities.h` for å gjennomføre den obligatoriske delen av denne øvingen. Siste deloppgave er frivilling og krever at du oppretter et grafikkprosjekt.

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 (15%)

a) **Definer en funksjon `inputIntegersUsingLoopAndPrintSum`**, som tar utgangspunkt i koden i funksjonen `inputIntegersAndPrintSum` fra oppgave 1. Den nye funksjonen skal la brukeren velge hvor mange tall som skal summeres, enten ved å angi antallet tall først, eller ved å slutte når brukeren skriver inn tallet 0.

b) **Teori:** Den forrige deloppgaven kan løses enten ved at brukeren angir antall tall som skal summeres i forkant, eller at man fortsetter å lese inn tall helt til brukeren skriver inn 0. Gjør kort rede for hvilken type løkke som er best egnet for hvert av alternativene.

c) **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.

d) **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.

e) **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 (25%)

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 Flere løkker (25%)

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. Alle beregninger kan gjøres med heltall.

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.

a) Serielån. Skriv funksjonen `calculateSeries` som skal regne ut årlige innbetalinger for et lån over et gitt antall år. 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: kostnaden av rentene som er påløpt og avdragsbeløpet som nedbetales og reduserer gjenstående gjeld.

Serielån har like store avdrag hver termin. Rentedelen av lånet vil variere, det betyr også at innbetalingsbeløpet vil variere. F.eks. vil 1000 kroner som skal betales ned over 10 år med 10% rente gi avdragsbeløp på 100 kroner hvert år og rentekostnaden i første termin er 100 kroner. Det blir totalt 200 kroner i første termin, men beløpet vil bli lavere for hvert år.

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. Funksjonen skal returnere en heltallsvector på samme måte som i forrige deloppgave.

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

d) Tegning av graf i FLTK. – Frivillig –
Hvis du har utført øving 0 og fått installert prosjekt-malen som følger dette faget og heter `TDT4102_Graphics` vil du kunne visualisere begge nedbetalingsplanene. Du må da opprette et nytt prosjekt av denne typen, kopiere over filene du allerede har laget ovenfor, samt ta inn den utdelte koden (`loan_drawer.h` og `loan_drawer.cpp`) som du finner under øving 2 på Blackboard. Disse filene beskriver en funksjon som gjør den nødvendige grafikk-koden for dette:

```
void drawPlot(vector<int> annuity, vector<int> series, int loan, int rate)
```

For å få tilgang til deklarasjonen av filen, må du inkludere den:
`#include "loan_drawer.h"`

Merk at det *ikke* er meningen at dere skal prøve å forstå den utdelte koden på dette stadiet i kurset, men at den bør være fullt forståelig noe senere i kurset. **Det er ikke nødvendig å få til dette for å få øvingen godkjent, men det er en god test på om du har**

fått installert FLTK riktig, og det vil du trenge i senere øvinger.

Bruk funksjonene fra deloppgave a) og b) til å lage to variabler av typen `vector<int>` som lagrer innbetalingene for henholdsvis annuitets og serielån. Kall funksjonen `drawPlot` med disse to som argumenter, i tillegg til heltallsargumentene for lånets størrelse og rente. (Merk at `drawPlot` er en veldig begrenset implementasjon fordi den ikke tar hensyn til om data skulle ligge utenfor forventet tall-område. Senere i kurset skal du lære hvordan det kan kompenseres for ved å skalere representasjonen etter variable data.)