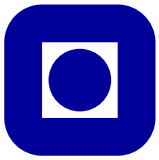

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2014

Øving 1

Frist: DD.MM.YYYY

Mål for denne øvinga:

- bli kjent med C++
- lære grunnleggende C++ og prosedyreorientert programmering
- lære hvordan programmer kan ta inn data og skrive ut på skjermen
- repetere grunnleggende programmering

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Det er valgfritt om du vil bruke en IDE (Visual Studio, XCode), men koden må være enkel å lese, kompilere og kjøre.

Anbefalt lesestoff:

- Kapittel 1 & 2, Absolute C++ (Walter Savitch)
- It's Learning notater

1 «Input/Output» og Funksjoner (15%)

a) Skriv en funksjon `getAndPrintInteger` som lar brukeren skrive inn et heltall (integer), og skriver dette tallet ut på skjermen

```
// 1a:
void getAndPrintInteger() {
    int tall = 0;
    cout << "Skriv inn et tall: ";
    cin >> tall;
    cout << "Du skrev: " << tall << endl;
}
```

b) Skriv en funksjon `getAndReturnInteger` som lar brukeren skrive inn et heltall (integer), og returnerer dette fra funksjonen, funksjonen skal *ikke* skrive tallet til skjerm

```
// 1b:
int getAndReturnInteger() {
    int tall = 0;
    cout << "Skriv inn et tall: ";
    cin >> tall;
    return tall;
}
```

c) Skriv en kommentar i koden din som forklarer hva slags returtyper disse funksjonene har, hvorfor de har dem, og forskjellen mellom disse returtypene. (minimum 5 setninger)

Svar:

`getAndPrintInteger` har void som returtype, siden den ikke skal returnere noe som helst. Funksjoner som ikke skal returnere noen verdi er typisk void. I dette tilfellet har vi ikke noe resultat, men skriver isteden noe til skjerm, et typisk eksempel på noe som kan være void.

`getAndReturnInteger` har int som returtype, siden den skal returnere et heltall (altså en int). Denne funksjonen kan ikke være void, ettersom vi er interessert i å få returnert verdien, det eneste andre alternativet her, er long, som det er lite sannsynlig at trengs her.

Forskjellen mellom int-retur og void-retur, ligger i at void-retur ikke returnerer noen verdi, mens int (eller en vilkårlig annen type retur) returnerer en verdi, som vi så kan bruke til å regne videre med. Dette kan sammenlignes med "Ans"-konseptet på kalkulatoren. Uten verdiretur ville vi måttet få verdiene skrevet inn igjen for hver gang vi trenger dem i en ny funksjon. På den annen side kan void være nyttig i tilfeller der en funksjon ikke resulterer i noen verdi. F.eks. vil en funksjon som skal printe noe til skjerm sjelden trenge å returnere noen verdi i tillegg.

d) Skriv en funksjon `getAndPrintSum` som ved å bruke en av funksjonene du nå har skrevet, tar inn to heltall og skriver ut kun summen på skjermen (*cout*).

```
void getAndPrintSum() {
    int number1 = getAndReturnInteger();
    int number2 = getAndReturnInteger();
    cout << "Summen av " << number1 << " og "
         << number2 << " er " << number1 + number2 << endl;
}
```

e) Skriv en kommentar i koden din som forklarer hvorfor du valgte å bruke den funksjonen du valgte (minimum 5 setninger)

Svar:

Her må vi bruke getAndReturnInteger(). getAndPrintSum() ville bare skrevet ut enkeltverdiene, uten å ta vare på dem på noen måte slik at vi kunne gjort noe mer med dem (som å summere dem, slik vi skal i denne oppgava). Dette demonstrerer dermed at det finnes tilfeller der vi trenger verdien som retur. Vi er heller ikke egentlig interessert i å skrive ut de verdiene vi nettopp skrev ut til skjerm ukritisk, vi vet tross alt hva vi nettopp skrev inn, og er mer interessert i å skrive ut summen av dem. Dermed vil verdiretur være riktig, og ikke verdiutskrift.

2 Løkker (10%)

a) Utvid funksjonen `getAndPrintSum` som du lagde i oppgave 1 til å la brukeren velge hvor mange tall som skal summeres, enten ved å angi antallet tall først, eller ved å slutte når brukeren gir tall med sum 0.

La den nye funksjonen hete `getAndPrintMoreSums`

```
// 2a - For-løkke-løsning
void getAndPrintMoreSums1() {
    int antallTall = 0;
    cout << "Hvor mange ganger vil du summere? ";
    cin >> antallTall;
    for (int i = 0; i < antallTall; i++) {
        int number1 = getAndReturnInteger();
        int number2 = getAndReturnInteger();
        cout << "Summen av " << number1 << " og "
             << number2 << " er " << number1 + number2 << endl;
    }
}

// 2a - While-løkke-løsning
void getAndPrintMoreSums2() {
    int sum = -1;
    cout << "Hvis summen er 0 avslutter vi,"
         << " alle andre tall fortsetter" << endl;
    while (sum != 0) {
        int number1 = getAndReturnInteger();
        int number2 = getAndReturnInteger();
        sum = number1 + number2;
        cout << "Summen av " << number1 << " og "
             << number2 << " er " << sum << endl;
    }
}
```

b) Skriv en kommentar i koden din som forklarer hvilken type løkke som er best egnet for hver av de to mulighetene i forrige oppgave, og hvorfor (minimum 5 setninger). **Svar:**

Hvis vi vet hvor mange ganger vi skal repetere noe, har vi typisk en for-løkke. I det første tilfellet i oppgave 2a har vi "angi antallet tall først" i oppgaveteksten, som betyr at antallet iterasjoner er kjent før vi begynner å iterere, ergo for-løkke. Hvis vi derimot ikke vet hvor mange ganger vi skal iterere, men heller avhenger av noe som skjer underveis for å finne antallet ganger, har vi typisk en while-løkke. While-løkke kommer som navnet hinter til av at vi skal gjøre noe "så lenge" noe er sant. I dette tilfellet skal summen ikke være 0, noe vi ikke kan vite hvor mange iterasjoner vi trenger for å oppnå før den iterasjonen hvor dette resultatet oppstår.

Det er verdt å nevne at ei for-løkke også kan skrives som en while, og vica-verca, så begge alternativene i 2a KAN teknisk sett løses med begge typer løkke. Det må dog nevnes at det blir litt mer kronglete, da for-løkker ikke egentlig egner seg veldig godt til tilfellet hvor antallet iterasjoner er helt ukjent, og while-løkker har ikke tellevariabler med i grunnlaget, slik at det blir litt merarbeid om man skal skrive ei telleløkke som while-løkke.

c) Skriv funksjonen `getAndReturnDouble` som skal gjøre det samme som `getAndReturnInteger`, men istedenfor å lese inn et heltall, skal denne funksjonen lese inn et desimaltall

```
// 2c:
double getAndReturnDouble() {
    double tall = 0;
    cout << "Skriv inn et tall: ";
    cin >> tall;
    return tall;
}
```

d) Skriv en funksjon som konverterer NOK til Euro.

```
// 2d:
void convertNOKtoEUR() {
    double NOK = -1.0f;
    while (NOK < 0.0f) {
        NOK = getAndReturnDouble();
    }
    double EUR = NOK * 7.84;
    cout << setprecision(2) << fixed
         << NOK << " NOK er " << EUR << " EUR" << endl;
    /** "Magic formula" page 31 in textbook -
        As an alternative to << setprecision(2) << fixed
      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(2);
    */
}
```

e) Skriv en kommentar i koden din som forklarer hvorfor vi ikke bør bruke `getAndReturnInteger` i forrige oppgave, men heller `getAndReturnDouble`, legg spesielt merke til hva i oppgaveteksten som legger føring på denne bruken. Kommenter også på returtypen til funksjonen du skrev i forrige oppgave. (Minimum 5 setninger).

Svar:

getAndReturnInteger gir oss et heltall, dette vil dermed ikke ha noen desimalplasser, som vil være problematisk når vi skal regne med valuta. Rett nok vil resultatet fortsatt bli regnet ut med riktig kurs, men øre-biten av verdien vil bli borte allerede FýR utregningen, noe som vil gi oss et svar som neppe er helt riktig. Når vi skal regne med desimaltall, bør vi derfor være nøye med å velge riktig datatype. I dette tilfellet er double tilstrekkelig, selv om double ikke er eksakt. Vi beholder dermed desimalplassene, og får et mer generelt korrekt svar fra utregningen. Det i oppgaven som legger føring på valg av datatype i dette tilfellet er ordleggingen "to desimaler", som gir oss et hint om at det her skal jobbes med desimaltall.

Returtypen til funksjonen er void, siden oppgaven sier "skriv ut det vekslede beløpet", og ikke "returner det vekslede beløpet".

f) Skriv om main() slik at brukeren kan velge i en meny mellom funksjonene fra foregående oppgaver

```
int main(int argc, char **argv) {
    int valg = -1;
    while (valg != 0) {
        cout << "0) Avslutt" << endl
             << "1) Summer to tall" << endl
             << "2) Summer flere tall (for)" << endl
             << "3) Summer flere tall (while)" << endl
             << "4) Konverter NOK til EURO" << endl
             << "5) Skriv ut gangetabell" << endl
             << "6) Konverter sekunder" << endl
             << "7) Regn ut tips og moms" << endl
             << "8) Sjekk om et tall er partall/oddetall" << endl
             << "9) Sjekk hvilket tall som er storst" << endl
             << "10) ABC-formelen" << endl
             << "11) Laanekalkulator" << endl
             << "Angi valg (0-11)";
        cin >> valg;
        cout << endl;
        switch (valg) {
        case 0:
            break;
        case 1:
            getAndPrintSum();
            break;
        case 2:
            getAndPrintMoreSums1();
            break;
        case 3:
            getAndPrintMoreSums2();
            break;
        case 4:
            convertNOKtoEUR();
            break;
        case 5:
            printMultiplicationTable();
            break;
        case 6:
            convertSecondsToTime();
            break;
        case 7:
            calculateVatAndTip();
            break;
        case 8:
            checkOddEven();
            break;
        case 9:
            printLargest();
            break;
        case 10:
            solveAndPrintRoots();
            break;
```

```
            case 11:
                calculateLoanPayments();
            default:
                cout << "Ugyldig valg" << endl;
            }
            cout << endl;
        }
    }
```

g) Skriv en funksjon som skriver ut en gangetabell på skjermen (cout). La brukeren gi både bredde og høyde på tabellen.

```
void printMultiplicationTable() {
    int height = 0;
    int width = 0;
    cout << "Hoyde?";
    cin >> height;
    cout << "Bredde?";
    cin >> width;
    for (int x = 1; x <= width; x++) {
        for (int y = 1; y <= height; y++) {
            cout << x *y << "\t";
        }
        cout << endl;
    }
}
```

Legg denne funksjonen til i testmenyen

3 Flere operatorer (10%)

a) Skriv en funksjon som lar brukeren skrive inn et antall sekunder (ved å bruke *cin*) og skriver ut (til cout) tilsvarende timer, minutter og sekunder.

```
// 3a
void convertSecondsToTime() {
    int seconds = 0;
    cout << "Skriv antall sekunder ";
    cin >> seconds;
    // Antall timer er sekunder / 3600
    // (desimaldelen forsvinner i int-matte)
    int hours = seconds / 3600;
    // La antall sekunder vaere resten fra divisjonen over.
    // Modulo (%) gir heltallsresten fra en divisjon,
    // f.eks. har 10/3, 3 som heltallssvar,
    // og 1 som rest (10 = 3 * 3 + 1).
    // Vi trenger ikke lengre alle sekundene som har gaatt opp
    // i hele timer.
    seconds = seconds % 3600;
    // Minuttene er da resterende sekunder / 60.
    int minutes = seconds / 60;
    // Og rene sekunder er resten fra denne divisjonen igjen
    seconds = seconds % 60;
    cout << hours << " time";
    if (hours != 1) {
```

```
        cout << "r";
    }
    cout << ", " << minutes << " minutt";
    if (minutes != 1) {
        cout << "er";
    }
    cout << " og " << seconds << " sekund";
    if (seconds != 1) {
        cout << "er";
    }
    cout << endl;
}
```

b) Skriv en funksjon som regner ut mva og tips for en resturantregning.

```
void calculateVatAndTip() {
    double price = 0.0f;
    cout << "Angi pris for maaltidet: " << endl;
    cin >> price;
    double vat = 0.0875 * price;
    double tip = (vat + price) * 0.18;
    cout << setprecision(2) << fixed
         << "Maaltid: " << price << endl
         << "MVA: $" << vat << endl
         << "Tip: $" << tip << endl
         << "----------------------" << endl
         << "Sum: $" << price + vat + tip << endl;
}
```

4 Kontrollstruktur (15%)

a) Skriv en funksjon som lar brukeren skrive inn et heltall og sjekker om tallet er et partall eller oddetall. La funksjonen skrive resultatet ut på skjermen (cout).

```
void checkOddEven() {
    int number = getAndReturnInteger();
    if (number % 2 == 1) {
        cout << "Tallet er et oddetall" << endl;
    } else {
        cout << "Tallet er et partall" << endl;
    }
}
```

b) Skriv en funksjon som lar brukeren skrive inn to desimaltall og skriver ut hvilket som er størst på skjermen (cout).

```
void printLargest() {
    double number1 = getAndReturnDouble();
    double number2 = getAndReturnDouble();
    if (number1 > number2) {
        cout << number1 << " er storst" << endl;
    } else if (number2 > number1) {
        cout << number2 << " er storst" << endl;
    } else {
        cout << "Tallene er like store" << endl;
    }
}
```

c) Skriv en kommentar i koden din hvor du forklarer hva som ville vært forskjellig dersom de foregående oppgavene hadde sagt «Skriv en funksjon som tar inn to desimaltall», istedenfor «Skriv en funksjon som lar brukeren skrive inn to desimaltall».

(Minimum 5 setninger)

Å forstå forskjellen mellom disse to konseptene er viktig fremover i øvingsopplegget.

Svar:

Hvis funksjonen hadde vært "tar inn", ville den tatt inn argumenter: printLargest(double number1, double number2). Disse ville måtte bli tatt inn et annet sted, og sendt inn som parametre til funksjonen. Dette kan sammenlignes med måten vi bruker matematiske funksjoner som sin(), cos() og tan() på: Når man bruker f.eks. tan(), angir man et parameter (i grader eller radianer), tan() spør ikke brukeren om input, men får input gitt.

Med andre ord ville funksjonen ikke hatt noe input internt, altså ingen cin-kall. Verdiene måtte isåfall bli lest inn i test-menyen, for så å sendes inn. Fordelen med dette er at en slik funksjon kan anvendes mer generelt. Hvis funksjonen i tillegg returnerer en verdi, istedenfor å printe den, kan den f.eks. brukes som en max()-funksjon.

5 Bruk av funksjoner i funksjoner, og røtter (25%)

I funksjonene under skal du tolke hva som skal være argumenter til funksjonen, og lære å bruke funksjoner i funksjoner. Generelt heretter i faget, skal funksjoner returnere verdier, og ikke skrive ut noe til skjerm, med mindre annet er spesifisert.

a) Skriv en funksjon `internalSum` som regner ut

$$b^2 - 4ac$$

og returnerer verdien (ingen utskrift til skjerm)

```
double internalSum(double a, double b, double c) {
    return pow(b, 2.0) - 4 * a * c; // Alternativt: return b*b - 4*a*c;
}
```

b) Skriv en funksjon `positiveSqrt` som regner ut

$$\sqrt{x}$$

dersom x er positiv eller 0, i alle andre tilfeller skal funksjonen returnere -1.

```
double positiveSqrt(double x) {
    if (x >= 0.0f) {
        return sqrt(x);
    }
    return -1;
}
```

c) Skriv en funksjon `polyRoot` som bruker de to foregående funksjonene til å regne ut

$$\sqrt{b^2 - 4ac}$$

```
double polyRoot(double a, double b, double c) {
    double root = positiveSqrt(internalSum(a, b, c));
    return root;
}
```

d) Skriv en funksjon `abcFormula` som regner ut løsning(ene) til

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

der a, b og c er gitt som argumenter til funksjonen, gjenbruk funksjonene fra forrige oppgave, og skriv ut resultatene til skjerm.

```
void abcFormula(double a, double b, double c) {
    double root = polyRoot(a, b, c);
    if (root > 0) { // 2 Løsninger
        double solution1 = (-b + root) / (2 * a);
        double solution2 = (-b - root) / (2 * a);
        cout << "Loesning 1: " << solution1 << endl;
        cout << "Loesning 2: " << solution2 << endl;
    } else if (root == 0) { // 1 Løsning
        double solution = (-b / (2 * a));
        cout << "Loesning 1: " << solution << endl;
    } else { // Ingen reell løsning
        cout << "Ingen reell loesning" << endl;
    }
}
```

e) **Lag en funksjon `solveAndPrintRoots` som lar brukeren skrive 3 desimaltall, bruk `abcFormula` til å regne ut røtten til andregradsuttrykket gitt ved disse tallene**

```
void solveAndPrintRoots() {
    double a = 0.0f;
    double b = 0.0f;
    double c = 0.0f;
    cout << "A: ";
    cin >> a;
    cout << "B: ";
    cin >> b;
    cout << "C: ";
    cin >> c;
    abcFormula(a, b, c);
}
```

f) **Legg til `solveAndPrintRoots` i testMenyen i `main()`**

g) **Buk testmenyen til å regne ut røttene til andregradsuttrykket gitt ved disse tallene. Test programmet med verdier som gir 0, 1, og 2 løsninger**

Hint: Prøv å regne ut røttene til:

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

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

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

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

6 Flere Løkker (25%)

a) Skriv en `calculateLoanPayments` funksjon som regner ut årlige innbetalinger av et lån over 10 år. Legg denne funksjonen til i testmenyen

```
void calculateLoanPayments() {
    double amount, interest;
    double total = 0;
    cout << "Laanesum: ";
    cin >> amount;
    cout << "Laanerente: ";
    cin >> interest;
    cout << "\t\tBetaling\tRest\n";
    for (int i = 0; i < 10; i++) {
        int payment = amount / 10 + (10 - i) / 10.0 * amount * (interest / 100.0);
        total += payment;
        cout << "Aar nr " << i + 1 << ":\t" << payment
             << "\t\t" << (9 - i) / 10.0 * amount << endl;
    }
    cout << "Totalt:\t\t" << total << endl;
}
```

b) Utvid funksjonen fra forrige deloppgave til å skrive ut oversikten over innbetalinger som en pen og pyntelig tabell, som ser ca slik ut:

År	Innbetaling	Gjenstående Lån
1	100000	100000000
2	90000	1000000

Hint: Bruk \t som tabulator i utskriften