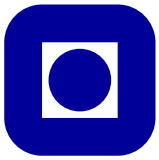

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2015

Øving 0.5-Matlab

Frist: 2015-01-16

Mål for denne øvinga:

- Overføre programmeringskonsepter vi kan fra Matlab til C++

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.

Denne øvingen er tenkt som ekstra hjelp til de av dere som har noe programmeringserfaring fra før men fra et annet språk en C++. Gjennom enkle eksempler ønsker vi å vise likhetstrekkene mellom det du kan fra før og C++. Øvingstempoet i faget blir ofte beskrevet som ganske høyt og denne øvingen kan være med på å gjøre oppstarten noe lettere og læringskurven mindre bratt.

I denne øvinga skal vi demonstrere likheter og ulikheter mellom C++ og Matlab, oppgavene vil i stor grad gå ut på å oversette Matlab-kode til C++, og kan kanskje være til hjelp som en slags oppfriskning av ITGK-kunnskapene, samtidig som de gir en mulighet til å overføre kunnskapen man har fra Matlab over til C++. Det vil også være viktig å legge merke til de fallgruvene og forskjellene som finnes mellom de to språkene.

Lynoppfriskning i Matlab, og C++ utgaven av tilsvarende kode

Generelt rammeverk

Siden vi ikke har gått så mye i dybden på C++ ennå, er det verdt å nevne enkelte ting som vi som regel trenger for å ha et gyldig program i C++. Et (nesten) minimalt C++-program, som ikke gjør noe som helst ser slik ut:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6         // Kode her
7         return 0;
8 }
```

En kort gjennomgang av hva som foregår her:

- Linje 1 bruker vi til å si at vi trenger `iostream`-biblioteket, som inneholder ting utskift til skjerm, og input fra tastatur, andre interessante bibliotek er f.eks. `cmath`, som inneholder sinus/cosinus/tanges, og kvadratrot/potensfunksjoner.
- Linje 3 angir at vi ønsker å bruke disse funksjonene uten å måtte skrive `std::` foran dem hver gang, denne linja vil du ha i (nesten) alle .cpp-filer.
- Linje 5 definerer `main`-funksjonen, det er her programmet ditt starter når du kjører det. Legg merke til krøllparentesene.
- Linje 6 inneholder her en kommentar. Disse starter med //, og sier at resten av linja skal ignoreres. Det er, som linja sier, typisk her man skriver koden sin.
- Linje 7 returnerer verdien 0 fra `main`, dette er konvensjon for å si at programmet fullførte uten feil.

Utskrift til skjerm

Matlab skriver ut til skjerm ved hjelp av «`disp`» og «`fprintf`». Analogt til dette bruker vi «`cout`» i C++ som følger:

```
disp(2+3)
disp('Hello world!')
fprintf('2+5=%d\n',2+5)
```

(a) Matlab

```
cout << 2+3 << endl;
cout << "Hello world!" << endl;
cout << "2+5=" << (2+5) << endl;
```

(b) C++

Figur 1: Utskrift til skjerm

I Matlab kan man skrive «`fprintf("A:%d B:%d C:%d", a,b,c)`», eller «disp(a, b, c)» i C++ bruker vi «`cout`» eller standard utstrøm. Hver enkel gjenstand man ønsker å skrive ut må sendes til «`cout`» ved hjelp av «<<»-operatoren. Det er tillatt å nøste denne / å skrive ut flere gjenstander av gangen slik som på siste linje i figuren. «`endl`» er definert på forhånd og symboliserer linjeskift (på samme måte som `'\n'` i `fprintf` i Matlab), dersom man ikke ønsker linjeskift etter utskrift kan denne droppes.

Variabler

I Matlab kan man introdusere variabelnavn ved å bare tilordne dem en verdi, i C++ må vi først definere hvilken type disse variablene er:

```
a = 1;
b = 2;
c = a + b;
d = c / b;
disp(d);
d = d * b;
disp(d);

e = c;
f = e / 2;
disp(f);
```

(a) Matlab

```
int a = 1;
int b = 2;
int c = a + b;
int d = c / b;
cout << d << endl;
d = d * b;
cout << d << endl;

double e = c;
double f = e / 2.0;
cout << f << endl;
```

(b) C++

Figur 2: Variabeldeklarasjon og bruk

Vi har her vist 2 forskjellige datatyper, double, og int, den øverste utregningen vil få en forkortningsfeil når vi bruker C++, siden int ikke inneholder et eneste desimal. Allikevel er int å foretrekke for utregninger som ikke trenger desimaler, siden double kan få avrundingsfeil. Legg også merke til at første gang en variabel brukes må vi skrive datatypen foran den. (Vi MÅ riktignok ikke tilordne den en verdi med en gang, men det er god skikk).

Input fra bruker

Matlab kan ta inn data ved hjelp av input(). I C++ bruker man isteden «`cin`», dette fungerer på en lignende måte som utskrift til skjerm.

```
i = input('Skriv inn et tall: ');
j = input('Skriv inn et tall: ');
fprintf('Summen av de to tallene: %d\n', i + j);
```

(a) Matlab

```
double i = 0.0;
double j = 0.0;
cout << "Skriv inn et tall:"<< endl;
cin >> i;
cout << "Skriv inn et tall:"<< endl;
cin >> j;
cout << "Summen av de to tallene: " << (i+j) << endl;
```

(b) C++

Figur 3: Input fra bruker

Matlab lar oss skrive ut en forespørsel til bruker om hva som skal skrives inn, mens vi i C++ vil måtte gjøre dette i to steg, først en utksrift, og så en forespørsel om input. C++ vil ved bruk av "»" operatoren ta seg av å lese inn og tolke verdien til riktig type. Tolkningen vil da gjøres, om mulig, til den datatypen variablen som brukes til lagring er av.

If-setninger

En If-test ser slik ut i Matlab og C++:

```
b = 2;
if b > 2
        disp('B is greater than 2');
else
        disp('B is less than or equal to 2');
end
```

(a) Matlab

```
int b = 2;
if (b > 2){
        cout << "B is greater than 2" << endl;
}
else{
        cout << "B is less than or equal to 2" << endl;
}
```

(b) C++

Figur 4: If-tester i Matlab og C++

De viktigste forskjellene for If-tester mellom Matlab og C++ er at Matlab ikke krever paranteser rundt det som skal testes og klammer ({}) brukes rundt koden som er i If-testen i C++.

For-løkker

En enkel for-løkke som kjører fra 1 til 10 ser slik ut i Matlab og C++:

```
for i = 1:10
    disp(i);
end
```

(a) Matlab

```
for (int i = 1; i <= 10; i++) {
    cout << i << endl;
}
```

(b) C++

Figur 5: Løkker i Matlab og C++

"for" ser litt annerledes ut, men vi finner igjen 1 og 10 som grenser også her, at vi har en steglengde på 1 er litt mindre intuitivt, men bakgrunnen for det ligger i "i++", som tilsvarer "i = i + 1", altså "øk i med 1". For et eksempel som skriver ut bare oddetallene (steglengde 2):

```
for i = 1:2:10
    disp(i)
end
```

(a) Matlab

```
for (int i = 1; i <= 10; i = i + 2) {
    cout << i << endl;
}
```

(b) C++

Figur 6: Løkker med steglengde i Matlab og C++

Som en notis: uttrykk på formen «`i = i + x`» skrives ofte som «`i += x`» i C++, i vårt tilfelle ville vi kunnet skrive «`i += 2`». Legg merke til at vi uttrykker lengden på for-løkka vår som en sammenligning «`i <= 10`». For ordens skyld er det normalt å starte løkkene på 0, og bruke «<» istedenfor «<=» i C++, blant annet fordi tabeller i C++ har første indeks på 0, og ikke på 1 som i Matlab.

While-løkker

```
i = 1;
while i < 1000
        i = i * 2;
        disp(i);
end
```

(a) Matlab

```
int i = 1;
while (i < 1000){
        i = i*2;
        cout << i << endl;
}
```

(b) C++

Figur 7: While løkker i Matlab og C++

Igjen er det kun mindre syntatiske (skrivemåte) forskjeller mellom en while-loop i Matlab og en i C++. Det er likevel greit å merke seg at C++ her også krever at det er paranteser rundt utrykket som testes og at istedenfor indentering bruker C++ krøllparanteser.

Tabeller

```
t = zeros(1,10);
for i = 1:10
    t(i) = i;
    disp(t(i));
end
```

(a) Matlab

```
int t[10];
for (int i = 0; i < 10; i++){
        t[i] = i;
        cout << t[i] << endl;
}
```

(b) C++

Figur 8: Tabeller i Matlab og C++

Tabeller opprettes litt anderledes i Matlab enn i C++. For å opprette en tabell i C++ er vi (for øyeblikket) avhengig av å vite den endelige størrelsen før programmet kompileres. Dette er fordi det må settes av nok minne til å holde tabellen. Når tabellen er opprettet vil den oppføre seg på nesten tilsvarende måte som i Matlab. Det er dog viktig å bite seg merke i forskjellene her:

- Tabell-indeksene starter på 0 i C++, dvs at en tabell på 10 elementer har indekser fra 0,1,...,9. Legg deg dette på minne, da det fort kan være litt uvant.
- Vi bruker [] og ikke () til å indeksere med i C++
- Tabellen har ikke noen veldefinert startverdi, hvis du ikke tilordner en verdi, vil det ligge "tilfeldige" verdier i cellene til tabellen, definert ut ifra hva som tilfeldigvis lå i minne der tabellen ble opprettet. Dette er ofte, men ikke alltid 0, så ting kan se pent ut når man kjører én gang, men bli feil neste om man ikke tilordner verdier.
- Tabeller kan i grunnlaget ikke endre størrelse eller vokse (men vi skal i løpet av faget lære andre løsninger som kommer rundt denne begrensningen)

Funksjoner

```
function [ result ] = getRandomNumber(seed)
    result = 4;
end
```

(a) Matlab

```
int getRandomNumber(int seed){
    return 4;
}
```

(b) C++

Figur 9: Funksjoner i Matlab og C++

Matlab skriver funksjoner som «function returverdi = funksjonsnavn(parameter)», istedenfor å skrive «function returVerdi» må vi i C++ definerer returtypen til funksjonen. Dette er variabeltypen til returverdien fra funksjonen. Returverdien i C++ har ikke noe navn som i matlab, men settes istedet av «return», som i tillegg avslutter funksjonen. Det er også viktig å nevne at alle argumenter må ha en type, i dette tilfellet tar funksjonen inn et heltall (int). Til slutt, funksjoner i C++ trenger ikke å lagres i en fil med samme navn som funksjonen.

Potens-operatoren

I Matlab bruker vi ^ for å opphøye et tall, i C++ finnes ikke denne, vi har to alternativer, for enkle uttrykk, som x^2, kan vi skrive det om som x * x, ellers kan vi bruke funksjonen `pow`, fra biblioteket `cmath`.

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main() {
        int fourSquared = pow(4,2);
        int fourCubed = pow(4,3);
        cout << "4^2: " << fourSquared << " 4^3: " << fourCubed << endl;
        return 0;
}
```

(a) Eksempel på potens

1 Kodeforståelse / oversett til Matlab (10%.)

a) Oversett følgende kodesnutter til Matlab.

```
bool isFibonacciNumber(int n){
    int a = 0;
    int b = 1;
    while (b < n){
        int temp = b;
        b = a + b;
        a = temp;
    }
    if (b == n){
        return true;
    }
    else{
        return false;
    }
}
```

Figur 11: C++ kode

2 Oversett fra Matlab til C++ (90%.)

Oversett følgende kodesnutter til C++ og sjekk at de kompilerer / kjører i ditt IDE.

a) Fibonaccirekker

```
function [ result ] = fibonacci( n )
    a = 0;
    b = 1;
    disp('Fibonacci numbers:')
    for x = 1:n
        fprintf('%d: %d\n', x, b)
        temp = b;
        b = a + b;
        a = temp;
    end
    disp('----');
    result = b;
end
```

Figur 12: Matlabkode

b) Trekanttall

```
function triangleNumbersBelow( n )
    acc = 1;
    num = 2;
    fprintf('Triangle numbers below %d: ', n);
    while acc + num < n
        acc = acc + num;
        num = num + 1;
        fprintf('%d ',acc);
    end
    fprintf('\n');
end
```

Figur 13: Matlabkode

```
function [ truthValue ] = isTriangleNumber( number )
    acc = 1;
    while number > 0
        number = number - acc;
        acc = acc + 1;
    end
    if number == 0
        truthValue = true;
    else
        truthValue = false;
    end
end
```

Figur 14: Matlabkode

c) Sum av kvadrerte tall

```
function [ totalSum ] = squareNumberSum( n )
    totalSum = 0;
    for i = 1:n
        totalSum = totalSum + i^2;
        disp(i^2);
    end
    disp(totalSum)
end
```

Figur 15: Matlabkode

d) Største av to tall

```
function [ largest ] = maxOfTwo( a, b )
    if a > b
        disp('A is greater than B');
        largest = a;
    else
        disp('B is greater than A');
        largest = b;
    end
end
```

Figur 16: Matlabkode

e) Primtall 1

```
function [ primeness ] = isPrime( n )
    primeness = true;
    for i = 2:(n-1)
        if mod(n,i) == 0
            primeness = false;
            break;
        end
    end
end
```

Figur 17: Matlabkode

f) Primtall 2

```
function naivePrimeNumberSearch( n )
    for number = 2:(n-1)
        if isPrime(number) == true
            fprintf('%d is a prime\n', number);
        end
    end
end
```

Figur 18: Matlabkode

g) Største fellesnevner

```
function [ greatestDivisor ] = findGreatestDivisor( n )
    for divisor = (n-1):-1:0
        if mod(n,divisor) == 0
            greatestDivisor = divisor
            break;
        end
    end
end
```

Figur 19: Matlabkode

h) Telling med lister

```
function [ noReturn ] = compareListOfNumbers( l )
    r = zeros(1,3);
    for i = 1:length(l)
        if l(i) < 0
            r(1) = r(1) + 1;
        elseif l(i) == 0
            r(2) = r(2) + 1;
        else
            r(3) = r(3) + 1;
        end
    end
    fprintf('%d numbers were below zero\n', r(1));
    fprintf('%d numbers were zero\n', r(2));
    fprintf('%d numbers were greater than zero\n', r(3));
end
```

Figur 20: Matlabkode