Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre- og objektorientert programmering Vår 2016

Øving 6

Frist: 2016-02-26

Mål for denne øvinga:

- Lære om operatorar og interaksjon mellom klasser av forskjellige typar.
- Lære å implementere og bruke klasser.

Generelle krav:

- Bruk dei eksakte namna og spesifikasjonane som er gitt i oppgåva.
- Det er valfritt om du vil bruke ein IDE (Visual Studio, Xcode), men koden må vere enkel å lese, kompilere og køyre.

Anbefalt lesestoff:

- Kapittel 5.4, 7, 8.1, 8.3,& 10.3 Absolute C++ (Walter Savitch)

Konvensjonar for klasser

I denne øvinga skal du implementere dine eigne typar ved å bruke klasser. Det er konvensjon i C++ at namna på typar (og dermed klasser) startar med stor forbokstav, og for å gjere koden meir leselig skal du følge denne konvensjonen.

Typenamna dine skal difor være på forma:

```
MyType
```

i motsetning til variabelnamn, som vanligvis er på forma

```
myVariable
```

og konstantar, som vanligvis har namn på forma

```
MYCONSTANT
```

Ein annan konvensjon du skal følge er at du skal lage ei headerfil (`.h`) og ei implementasjonsfil (`.cpp`) for kvar klasse. Hvis klassen heiter `Car`, skal du altså lage filene `Car.cpp` og `Car.h`, og desse skal innehalde all kode for klassen `Car`.

1 Ei enkel klasse for 2×2-matriser (20%)

Ei matrise er eit matematisk omgrep frå lineær algebra som har applikasjonar i dei fleste vitskapar.

Reint praktisk er ei matrise eit strukturert rutenett av tal som har eit sett reglar om korleis matriser samhandlar med kvarandre. I denne delen av øvinga skal vi sjå på den enklaste formen for matriser, 2×2-matriser. I dei påfølgande delane skal vi óg kike på korleis enkle matriser og vektorar samhandlar.

Som du allereie har lært, lagar vi nye datatypar i C++ ved å deklarere klasser. Difor ynskjer vi å lage ei klasse for å representere matrisar i programma våre.

a) Lag ei klasse kalla `Matrix2x2`, som skal innehalde lagringsplass for 4 tal av typen `double`.

Når du vel korleis du skal lagre data er det ofte nyttig å vite korleis dei kjem til å verte nytta seinare. I matriseoperasjonar kjem vi til å aksessere dei forskjellege elementa i matrisa basert på rada og/eller kolonna deira. Difor er det nyttig å ordne medlemmane i `Matrix2x2` som eit todimensjonalt array.

b) Lag `set`- og `get`-funksjonar for matrisa.

```
double get(int row, int column) const;
void set(int row, int column, double value);
```

Om du ikkje har gjort det allereie, forsikre deg om at medlemsvariablane dine er private og at get/set-funksjonane er public.

c) Test funksjonane dine.

Lag eit main-program, lag ei matrise, og skriv ut verdiane til matrisa.

Tips: Det er antagelegvis nyttig å lage ein print-funksjon for å vise elementa i matrisa på ein strukturert måte. Denne funksjonen kan vere anten ein medlemsfunksjon eller ein normal funksjon.

d) Lag ein standardkonstruktør (default constructor).

Som du kanskje har lagt merke til i den forrige oppgåva, får du søppeldata ut om du prøver å skrive ut ei matrise der elementa ikkje har vorte eksplisitt tilordna.

Måten å løyse dette problemet på er å eksplisitt lage ein standardkonstruktør. Den skal sette elementa i matrisa til å vere lik *identitetsmatrisa.* Identitetsmatrisa er ei matrise som har 1 langs diagonalen ifrå øvre-venstre til nedre-høgre, og 0 ellers.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

e) Lag ein hjelpekonstruktør.

Det vert fort eit irritasjonsmoment å måtte skrive fire linjer kode kvar gong du ynskjer å lage ei ny matrise. Som programmerar bør du styre unna ekstraarbeid av fleire årsaker... Lag difor ein hjelpekonstruktør som tek alle fire elementa til matrisa som parametrar, og sett dei på riktig plass.

2 Overlaste operatorar (30%)

I denne oppgåva skal vi legge til nokre overlasta operatorar, slik at vi kan skrive enkle uttrykk med matrisene våre på ei kortare og meir lesbar form.

a) Overlast `operator <<`, som skal skrive ut innhaldet i matrisa til ein `ostream`.

Merk at ettersom operatoren tar inn ein `ostream` som første argument, kan den ikkje implementerast som ein medlemsfunksjon.

Tips: Å overlaste denne operatoren tidlig vil gjere livet ditt enklare når du skal teste ei klasse du lagar. **Drøft:**

- Kvifor bør denne funksjonen ta `const` argument? (t.d. `const Matrix2x2 &rhs`)
- Når bør vi, og når bør vi ikkje (evt kan ikkje) nytte `const`-argument?

b) Overlast `operator +=` og `operator -=`.

Summen av to matriser er den elementvise summen.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

Liknande når ein trekker to matrisar frå kvarandre, gjer vi ein elementvis subtraksjon.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a-e & b-f \\ c-g & d-h \end{bmatrix}$$

Reflekter over følgjande:

- Kvifor kan vi *ikkje* berre skrive:
  ```
  *this = *this + rhs;
  ```
- Kvifor kan du nytte operatorane += og -= når du arbeider med *elementa* i matrisa?

c) Opprett følgjande matriser i `main`:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},\ B = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix},\ C = \begin{bmatrix} 1.0 & 3.0 \\ 1.5 & 2.0 \end{bmatrix}$$

Test følgjande uttrykk ved å nytte += og -=, skriv ut begge resultata:

```
A += B
A -= B
```

Endre A tilbake til sin opprinnelege verdi etter testen.

d) Overlast +, -, og * operatorane. Produktet av to 2x2 matrisar kan reknast ut slik:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} (a*e+b*g) & (a*f+b*h) \\ (c*e+d*g) & (c*f+d*h) \end{bmatrix}$$

Tips:

- *Når du overlastar `+` og `-`, kan du nytte dei allereie overlasta operatorane `+=` og `-=`. Å unngå duplisering av kode er som regel ein god ting. Å gå andre vegen er også ei moglegheit, men vi valte her å implementere `+=` og `-=` fyrst.*
- *Produktet av to matriser kan skrivast som ei trippel for-løkke. (Noko som kan vere nyttig å kunne til Øving 6)*

e) Test implementasjonen din på matrisene vi lagde i c med følgjande uttrykk:

```
D = A+B
D = D*B - A + C
```

og skriv ut resultatet.

f) Overlast `operator *=`.

Tenk over følgjande:

- Kvifor kan du ikkje implementere *= på same måte som vi implementerte += og -= tidlegare?
- Kan du implementere *= ved å gjenbruke kode du har skrive allereie?

g) Skriv om testen fra e) til å nytte operatoren `*=`, slik at du også testar `*=`.

3 Samspel mellom vektorar og matriser (15%)

I lineær algebra er det mykje som er likt ved matrisar og vektorar. Eksempelvis kan ein behandle ein vektor som ei matrise med berre éi rad eller berre éi kolonne. Dette gjer det mogleg for oss å definere matrise-produktet mellom ei matrise og ein kolonnevektor slik:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} a*e+b*f \\ c*e+d*f \end{bmatrix}$$

a) Lag ei ny klasse `Vector2` som held ein matematisk vektor med to koordinatar.

NB: Det er her snakk om ein matematisk vektor, og *ikkje* `std::vector` frå standardbiblioteket til C++.

Forsikre deg om at designet ditt følgjer OOP-konvensjonane angåande innkapsling (encapsulation), og skriv nødvendige/passande get- og set-funksjonar.

b) Implementer følgjande medlemsfunksjonar:

- **Returnerer dot-produktet mellom this og rightHandSide**

```
double dot(const Vector2 &rightHandSide) const;
```

- **Returnerer den kvadrerte lengda av vektoren**

```
double lengthSquared() const;
```

- **Returnerer lengda av vektoren. Kvadratrota av verdien kan reknast ut med funksjonen `sqrt(double)` som finst i `<cmath>`-headeren.**

```
double length() const;
```

Hint: Du kan gjenbruke dot-funksjonen for å rekne ut ***`lengthSquared`****, og gjenbruke* ***`lengthSquared`*** *for å rekne ut* ***`length()`****.*

Drøft:

- Kva betyr `const` som står bak disse funksjonshovuda?
- Når bør ein nytte `const` på denne måten?

c) Implementer `operator` << for `Vector2`.

d) (Valfri) **Implementer binæroperatorane `+=`, `-=`, `+` og `-` og unæroperatoren `-` for klassa `Vector2`.**

Medan desse operatorane er nødvendige for en funksjonelt komplett implementasjon av ein vektor, vil vi sjå seinare i kurset at disse operatorane kan verte overflødige ved å bruke arv.

4 Meir matrisefunksjonalitet (25%)

For å kunne gjere noko nyttig med matrisene og vektorane våre må vi implementere matrise-vektor-produktet, i tillegg til nokre få nyttige funksjonar i `Matrix2x2`.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{1}$$

a) Lag medlemsfunksjonen `det()`, som reknar ut determinanten til matrisa.
Prototype:

```
double det() const;
```

Determinanten til matrise A er gitt som følgjer:

$$\det(A) = a * d - b * c$$

b) Lag medlemsfunksjonen `inverse()`.
Denne funksjonen skal returnere inversen av matrisa (om ein slik finst). Prototype:

```
Matrix2x2 inverse() const;
```

Inversen av ei 2×2-matrise på forma (1) er gitt som følgjer:

$$\text{inv}(A) = \frac{1}{\det(A)} * \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Tips: Legg merkje til at inversen av ei matrise ikkje nødvendigvis eksisterer. Ei matrise er kun inverterbar dersom determinanten er ulik 0. ***Om determinanten er 0 må vi ha ein måte å flagge dette på****. Det er fleire måtar å løyse dette på:*

- *Vel ein ugyldig verdi som skal returnerast ved feilaktige matriser. (Eit døme er null-matrisa).*
- *Endre prototypen for* ***inverse*** *slik at den tek resultatmatrisa som parameter (call-by-reference), og returnerer ein bool. Du kan då nytte den boolske verdien til å avgjere om matrisa vart invertert på riktig måte.*
- *Nytte* ***throw*** *og* ***exception****, som dekkjast seinare i kurset*

Korleis du vel å løyse dette, er opp til deg.

c) Implementer `operator *` mellom `Matrix2x2` og `Vector2`. Implementer matrise-vektor-produktet som vist i Del 3, resultatet skal vere ein Vector2.

Tenk over følgjande:

- Kvifor gir det inga meining å implementere operatoren `*=` i dette tilfellet?

5 Sy alt saman (10%)

Eit system av lineære likningar kan uttrykjast som ei matriselikning. Dette er måten eit system av likningar med hundrevis (eller til og med millionar) av ukjente løysast.

Gitt eit system av likningar med to ukjente (anta a, b, c, d, p og q som gitt):

$$\text{I} : ax + by = p$$
$$\text{II} : cx + dy = q$$

Den korresponderande likninga er:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}$$

Der målet er å avgjere kva x og y er.

Ein måte å løyse dette systemet er å multiplisere båe sider av likninga med inversen av matrisa $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, som gir:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \text{inv} \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} p \\ q \end{bmatrix}$$

Oppgåva di no er å kombinere dette til eit program som tek inn a, b, c, d, p og q frå brukaren og skriv ut dei resulterande x og y, eller ei feilmelding som seier at løysinga ikkje eksisterer.