Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2014

Øving 2

Frist: DD.MM.YYYY

Mål for denne øvingen:

- Lære om funksjoner
- Lære forskjellene mellom vanlige variabler og pekere
- Lære å skrive koden din i flere filer
- Lære å bruke nyttig matematiske funksjoner som sin og cos

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner som er 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
- Dersom noe er uklart eller trenger en bedre forklaring ta kontakt med en studass på sal.

Anbefalt lesestoff:

- Kapittel 3 & 4, Absolute C++ (Walter Savitch)

Når man skriver et C++ program er det viktig at man er strukturert. Vanligvis vil et program bestå av minst følgende:

- En "main" fil
- En ".h" fil (Header-fil)
- En ".cpp" fil (Implementasjons-fil)

Hoved / "main" filen

Dette er filen som kjøres når programmet startes. Utførelsen av programmet vil begynne i en funksjon som heter main. Dette funksjonsnavnet er standard for alle C++ programmer.

Header-filen

Denne filen inneholder informasjon som gjør det mulig å bruke den koden du skriver uten å vite hvordan den egentlig fungerer. Kan sees på som en enkel brukermanual for programmet.

Filen inneholder normalt funksjonshodene til alle funksjoner som skal være tilgjengelig/mulig å bruke for andre / utenforstående kode.

Implementasjons-filen

I denne filen skal resten av koden ligge. Dette er vanligvis der man skriver mesteparten av koden. I motsetning til header filen som kun sier noe om hvordan man kan bruke koden, vil denne filen beskrive implementasjonen.

Filenes relasjon til hverandre:

Header filen skal inkluderes av både "main"-filen og implementasjons-filen. Dette gjør at funksjonene som er beskrevet er tilgjengelige fra "main"-funksjonen og det sørger også for at koden i implementasjonsfilen blir ryddigere (vi kommer tilbake til dette).

Oppgaveforklaring:

I denne øvingen kommer vi til å se på gjenstander i bevegelse. Spesifikt ønsker vi å se på banen til en kanonkule som blir skutt ut med en gitt vinkel og fart. Vi kommer til å prøve å forutsi denne banen på to måter:

- Direkte med formel
- Ved numerisk integrasjon

En del synes integraler er vanskelig, men ikke la deg stoppe av det! Tanken bak øvingen er at dette kan gi et eksempel der datamaskiner kan gjøre oppgaver som er langtekkelig for mennesker men som kun tar en datamaskin et par sekunder.

Alle formlene og uttrykkene som brukes kan utledes på egenhånd, men da dette er ikke et fag i numerisk løsning av integraler kommer dere til å få alle uttrykk og formler som trengs for å løse disse oppgavene. Oppgaven består av å å sette dette sammen til et fungerende C++ program. Det er ikke forventet at dere skal trenge å bruke mye tid på å forstå problemet og sette opp ligninger for å løse det.

Del 1: Enkel bevegelse:

1 Funksjonshoder (10%)

I denne oppgaven skal vi lage et sett med funksjonshoder. Funksjonshodene vil bli beskrevet med tekst og det er opp til deg å tolke dette og skrive passende funksjonshoder. Alle funksjonshoder skal være i en egen "header"-fil. I denne øvingen skal den hete `"cannonball.h"`. Det er vanlig at man gruperer funksjoner og funksjonalitet som hører sammen i biblioteker som vi kan bruke senere uten å vite hvordan de fungerer internt.

De påfølgende funksjonshodene danner grunnlaget for et bibliotek som kan regne ut banen til en kanonkule. Oppgaven er strengt spesifisert og det skal ikke være nødvendig å gjøre antagelser. Om du likevel føler at det er noe som er tvetydig eller dårlig spesifisert noter gjerne dette i koden din (kommentar i C++ kan skrives som /* */) og ta det opp med din studentassistent på sal.

Eksempel-deloppgave:

Denne første deloppgaven skal ikke gjøres, kun kopieres til din header fil da du kan få bruk for den i senere oppgaver.

a) Lag et funksjons hode som returnerer akselerasjonen i y-retning (oppover). Akselerasjonen i y-retning er normalt et desimaltall. Denne funksjonen skal hete "acclY".

Løsning:

```
double acclY();
```

Husk at ingen av de påfølgende deloppgavene forventer noe mer en bare funksjonshodet som i eksempelet over.

b) Skriv et funksjonshode som regner ut farten i y-retning (oppover). Denne funksjonen tar inn to flyttall (double): startfart og tid. Til slutt returnerer funksjonen farten som et flyttall. Funksjonen skal hete `"velY"`.

c) Skriv et funksjonshode som regner ut farten i y-retning (oppover) med integrasjon. Denne funksjonen tar inn to flyttall (double): startfart og tid. Til slutt returnerer funksjonen farten som et flyttall. Funksjonen skal hete "velIntY".

d) I denne deloppgaven skal vi lage et sett av funksjonshoder for utregning av posisjon både i X- og Y-retning. Det skal være en funksjon for hver retning samt hver måte å regne ut svaret på (formel og integrasjon) dette gir 4 funksjonshoder:

Posisjon i X-retning med formel
Posisjon i X-retning med integral
Posisjon i Y-retning med formel
Posisjon i Y-retning med integral

Alle disse tar inn to flyttall hver: startfart og tid. Vi går ut i fra at start-posisjonen er 0 (så denne trenger vi ikke å ta inn).

e) Skriv et funksjonshode som tar inn tid i sekunder og ikke returnerer noe. Denne skal hete `"printTime"`.

f) Skriv et funksjonshode som tar inn startfarten i y-retning og returnere flytiden i sekunder. Denne skal hete `"flightTime"`.

2 Implementer funksjoner (20%)

I denne oppgaven skal vi implementere funksjonene fra forrige oppgave. Alle funksjonsimplementasjoner skal ligge i en implementasjonsfil tilhørende "header"-filen. I dette tilfellet heter den `"cannonball.cpp"` da vi skal implementere funksjonenen som er lagt i `"cannonball.h"`.

Formler som trengs vil bli presentert. Din oppgave er å skrive en funksjon som bruker formlen og passer til funksjonshodet som er laget tidligere. Det kan lønne seg å kopiere funksjonshodene fra "header"-filen som en start da funksjonene må passe funksjonshodene eksakt for at det skal fungere. Husk også å inkludere "header"-filen i implementasjons-filen, dette kan gjøres ved:

```
#include "cannonball.h"
```

Det er viktig at begge filene, "cannonball.h" og "cannonball.cpp" ligger i samme mappe for at dette skal fungere.

a) I denne oppgaven skal du implementere funksjonen fra oppgave 1a. Denne funksjonen returnerer akselerasjonen i y-retning (oppover). Til vanlig er akelerasjonen i y-retning -9.81m/s (gjenstander trekkes mot bakken).

b) Denne oppgaven skal implementere funksjonen fra oppgave 1b (fart i y-retning). Funksjonen gjør følgende utregning basert på de verdiene den får inn:

$$fartY = startFartY + akselY * tid \tag{1}$$

c) Nå skal vi implementere funksjonen fra oppgave 1c (fart i y-retning med integral). Det kan skrives på følgende måte:

$$fartY = startFartY \tag{2}$$

$$fartY = fartY + akselY * liteTidsSteg \tag{3}$$

Forskjellen her er at vi gjøre dette mange ganger. Vi deler det hele opp som følger:

$$antallSteg = tid/liteTidsSteg \tag{4}$$

For hvert steg må vi legge til litt til den totale farten (som i **??**).

Hint: en løkke av et eller annet slag er nødvendig.

d) I denne oppgaven skal vi lage / implementere funksjonene som regner ut posisjonen i X- og Y-retning (oppgave 1d). Formlen for dette er:

$$posisjon = startPosisjon + startFart * t + \frac{1}{2} * aksel * t^2 \tag{5}$$

For å gjøre tilsvarende med integrasjon kan vi gjøre følgende:

$$posisjon = startPosisjon \tag{6}$$

$$posisjon = posisjon + fartNå * liteTidsSteg \tag{7}$$

$$antallSteg = tid/liteTidsSteg \tag{8}$$

Det er opp til deg å finne ut hva som skal hvor, men et tips kan være å benytte funksjoner som du har laget tidligere samt å se nøye på integrasjon av fart for et eksempel på hvordan integrasjon kan gjøres (forrige oppgave). Posisjonen i X-retning er ikke avhengig av posisjon i Y-retning og motsatt. De kan regnes ut hver for seg gitt tid og startfart.

e) Implementer funksjonen `"printTime"` (oppgave 1e). Vi ønsker å dele opp sekundene i timer, minutter og sekunder (sekunder kan evt være et desimaltall) for så å skrive dette til skjerm.

Hint: se hvor mange timer det er først.

f) Implementer funksjonen `"flightTime"` (oppgave 1f).

`"FlightTime"` skal finne ut hvor lenge noe kommer til å fly (i dette tilfelle kanonkulen) gitt en fart. Det er kun farten i Y-retning som har noe å si siden vi går ut ifra en perfekt flat bakke og ingen luftmotstand etc. Forestill deg følgende: en kule som kastes rett opp mister etter hvert farten på grunn av tyngdekraften. Når den er på det høyeste vil farten være null, deretter vil den begynne å falle. Når kulen igjen er ved bakken vil farten være like stor som den var ved starten av kastet, men motsatt rettet (altså nedover mot bakken).

3 Verifiser at funksjonene fungerer (10%)

Det er veldig lurt å teste koden litt etter litt. Ofte kan det være utfordrende å finne feil dersom man kjører større programmer sammen da man ikke vet hvor feilen oppstår. Når man tester C++ kode ønsker vi å finne ut følgende:

- Kompilerer programmet? Dette luker ut de fleste syntaksfeil og grove logiske feil.
- Gjør programmet det du forventer at det skal gjøre? For å teste dette må vi vite noe om hva programmet skal gjøre dersom det fungerer.

a) Forsikre deg om at programmet kompilerer.

Dersom programmet ikke kompilerer sjekk følgende:

- Programmet har følgende filer: `"main.cpp"`, `"cannonball.h"` og `"cannonball.cpp"`
- Filene er satt opp som følger: `"main.cpp"` inkluderer `"cannonball.h"` og `"cannonball.cpp"` inkluderer `"cannonball.h"`
- I filen `"main.cpp"` finnes det en main funksjon (`"int main()"`)

Dersom det fortsatt ikke fungerer når du prøver å kompilere må du se på feilmeldingene. Dersom du ikke forstår feilmeldingen kan Google og din studentassistent være til hjelp.

b) Test hver metode fra main funksjon.

Hver metode bør testes, dette kan gjøres enkelt ved å bruke et sett av eksempeldata der man vet svaret på forhånd. For eksempelverdier se tabellen under:

	T = 0	T = 2.5	T = 5.0
acclX	0	0	0
acclY	-9.81	-9.81	-9.81
velX	50.0	50.0	50.0
velY	25.0	0.475	-24.05
posX	0.0	125.0	250.0
posY	0.0	31.84	2.375

I dette tilfellet kommer de fleste av testene til å involvere desimaltall. Desimaltall fungerer noe anderledes i en datamaskin enn det som er naturlig, dette kan du lese mer om på Wikipedia:

`http://en.wikipedia.org/wiki/Floating_point#Accuracy_problems`

For denne oppgaven holder det å si at det å sammenligne (dnumber1 == dnumber2) er en dårlig ide. Det er to grunner til dette:

1. Det finnes tilfeller der to tilsynelatende like tall (om du skriver de til skjerm så ser de like ut) ikke er representert på samme måte i datamaskinen og dermed vil bli sagt å være ulike! Dette er på grunn av avrunding.
2. På grunn av forskjeller i datamaskinens prosessor er det ikke garantert at en lik sammenligning vil returnere rett selv om de to tallene er like (fordi datamaskinen internt kan velge å lagre tallene på forskjellige måter).

Den beste måten å sjekke om svaret stemmer på vil være å teste om avviket fra det korrekte svaret et veldig lite, f.eks:

```
double svar = /*svaret settes her */
double avvik = pow(fasit-svar,2.0);
if (avvik < feilmargin){
        /* Fantastisk, det var rett! */
}
else{
        /* Svaret var galt, det falt utenfor feilmarginen */
}
```

Del 2: Gjenbruk av funksjoner

4 Implementer funksjoner (20%)

a) I denne oppgaven skal vi implementere følgende funksjoner:

```
void getUserInput(double *theta, double *absVelocity);
double getVelocityX(double theta, double absVelocity);
double getVelocityY(double theta, double absVelocity);
void getVelocityVector(double theta, double absVelocity,
        double *velocityX, double *velocityY);
```

Disse funksjonene brukes til å lese inn en vinkel og en fart fra brukeren. For å kunne returnere mer enn en verdi kan vi benytte oss av pekere. Pekere er som navnet tilsier en peker til en plass i minnet til datamaskinen. Fordelen med å jobbe med pekere er at funksjonen som får pekeren har mulighet til å gå inn i minnet og endre verdien slik at endringen også blir gjeldende utenfor funksjonen. Syntaks for pekere som funksjonsargument kan sees i `"getUserInput"`. `"double *"` betyr her at denne tar inn en peker som peker til en plass i minnet der det finnes en "double"-variabel.

Når vi bruker pekere må vi skille mellom å endre minneaddressen som den peker til og innholdet som blir pekt på, for å endre innholdet som pekes på gjør man følgende:

```
void foo(double *b){
        *b = 2.0;
}
```

For å endre minneaddressen som blir pekt på gjør man dette (eksempelet nedenfor setter pekeren til NULL): I dette eksempelet vil adressen bli kopiert til en ny lokal variabel `"b"`.

```
void foo(double *b){
        b = NULL;
}
```

Det gjør at endringer på `"b"`, det vil si **adressen** `"b"` peker til, vil ikke få noen effekt utenfor funksjonen! Funksjonen over gjør ingenting!

Ser man bort ifra at man må skrive *b istedenfor b, kan pekere brukes på samme måte som vanlige variabler. Når vi skal kjøre funksjonen som tar inn en peker kan vi gjøre følgende:

```
double a;
foo(&a);
```

Dette gjør at vi sender inn addressen der "a" er lagret fremfor verdien "a" har.

Funksjonen "getVelocityX" skal dekomponere absoluttfarten i X-retning, dette kan gjøres som følger:

$$fartX = absFart * \cos(vinkel) \tag{9}$$

Tilsvarende gjøres for "getVelocityY":

$$fartY = absFart * \sin(vinkel) \tag{10}$$

Funksjonen "getVelocityVector" kjører begge funksjonene ("getVelocityX" og "getVelocityY") og returnere "velocityX" og "velocityY" gjennom pekere.

b) Implementer funksjonen "getDistanceTraveled"

```
double getDistanceTraveled(double velocityX, double velocityY);
```

Funksjonen "getDistanceTraveled" skal returnere avstanden som kanonkulen greide å traversere før den traff bakken, med andre ord hva "Posisjon X" er når "Posisjon Y" er 0. Den enkleste måten å løse dette på er ved å bruke funksjonen "flightTime".

c) Implementer funksjonen "optimalAngleForMaxDistance"

```
double optimalAngleForMaxDistance(double absVelocity);
```

"optimalAngleForMaxDistance" skal returnere hvilken vinkel som gjør at kanonkulen fløy lengst. Dette kan gjøres ved hjelp av "getDistanceTraveled" og få datamaskinen til å prøve forskjellige vinkler.

d) Implementer funksjonen `"targetPractice"`. Funksjonen skal ta inn en avstand `"distanceToTarge` og returnere avvik fra denne dersom `"velocityX"` og `"velocityY"` er henholdsvis startfart i X- og Y-retning.

```
double targetPractice(double distanceToTarget,
        double velocityX, double velocityY);
```

5 Verifiser at funksjonene fungerer (10%)

a) Verifiser at programmet kompilerer

Tips: Har du lagt til funksjonshodene i "header"-filen?

b) Lag en kodesnutt i `"main()"` som tester funksjonene

6 Større program (20%)

a) Legg til funksjonen `"playTargetPractice"` til biblioteket og legg til kode i `"main()"` for å kjøre denne funksjonen.

```
void playTargetPractice();
```

Vi ønsker i denne oppgaven å sette sammen alle funksjonene til et større program. Dette programmet skal ta inn en vinkel og fart fra brukeren, deretter skyte med kanonene mot et tilfeldig plassert mål og si hvor langt unna brukeren var fra å treffe målet. Spilleren skal få 10 forsøk på å treffe målet og for hvert forsøk skal marginen samt om skuddet var for langt eller for kort skrives til skjerm.