



Norges teknisk-naturvitenskapelige
universitet
Institutt for datateknologi og
informatikk

TDT4102 Prosedyre-
og objektorientert
programmering
Vår 2024

Øving 6

Frist: 2025-02-21

Mål for denne øvingen:

- Lese fra og skrive til filer
- Strømmer (streams)
- Assosiative tabeller (map)
- Operatoroverlastning
- Animasjon

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal leveres på INGIInious.
- Benytt VS Code til å skrive, kompilere og kjøre kode.

Anbefalt lesestoff:

- Kapittel 8.6, 9.3, 14.6 og 20.2 i PPP
- [Dokumentasjon](#) til AnimationWindow -> Animasjon.

1 Lese fra og skrive til fil (20%)

Nyttig å vite: Filhåndtering

Dersom man skal skrive til eller lese fra filer ved hjelp av C++ kan man ta i bruk `std::ifstream` og `std::ofstream`. Disse må inkluderes fra standardbiblioteket med headeren `<fstream>`.

`std::ofstream` (Output File Stream) brukes når man skal skrive data til en fil. Når datatypen er konstruert kan den brukes på samme måte som `std::cout`, som vist under.

```
//variabel som holder filstien til myFile.txt
std::filesystem::path fileName{"myFile.txt"};
//variabel som holder en strøm til filen myFile.txt
std::ofstream outputStream{fileName};

outputStream << "This file contains some text" << std::endl;

int number = 10;
outputStream << "You can also output integers: " << number << std::endl;

double anotherNumber = 15.2;
outputStream << "Or floating point numbers: " << anotherNumber << std::endl;
```

`std::ifstream` (Input File Stream) brukes når man skal lese fra en fil. Når datatypen er konstruert kan den bruke på samme måte som `std::cin`. Dersom man ønsker å lese hele filen ord for ord kan man gjøre dette med en while-løkke, som vist under.

```
std::ifstream inputStream{"myFile.txt"};

if (!inputStream) { // Sjekker om strømmen ble åpnet
    std::cout << "Could not open file" << std::endl;
}

std::string nextWord;

while(inputStream >> nextWord) {
    std::cout << nextWord << std::endl;
}
```

Man kan også lese filen tegn for tegn ved å bruke en `char` i stedet for en `std::string`. Om en ønsker å lese filen linje for linje kan man bruke `std::getline`, denne funksjonen er beskrevet i infobanken til *Cin*.

Du kan også lese om hvordan man leser fra fil, og skriver til fil i kapittel 9.3 i læreboken.

a) Les ord fra brukeren og skriv de til en fil.

Implementer funksjonen `writeUserInputToFile` i `FileUtils.cpp` som lar brukeren skrive inn ord på tastaturet (`cin`) og lagrer hvert ord på en separat linje i en tekstfil. Lagre hvert ord i en `string` før du skriver det til filen og la ordet «quit», avslutte programmet. Det er viktig at akkurat ordet `quit` er det som avslutter programmet, ellers fugerer ikke autoretteren.

b) Legg til linjenummer.

Implementer funksjonen `addLineNumbers` som leser fra en tekstfil, og lager en ny fil med den samme teksten, der hver linje har linjenummer som første tegn. Linjenummerne skal

telle fra og med 1. Den nye filen skal ekstakt samme navn som den forrige, men med .linum på slutten. F.eks. hvis inputfilen er «file.txt», skal outputfilen hete «file.txt.linum». Sørg for at programmet ditt sjekker at filen eksisterer. *Hint: å lese en hel linje av gangen sparer mye arbeid og viser tydelig hva intensjonen er, framfor å lese en linje ord- eller tegnvis og sjekke for linjeskift.* Se kap. 9.3.

Nyttig å vite: filstier

Filstier forklarer hvor på datamaskinen noe er lagret. Ta for eksempel `Øving6.pdf`. Når du allerede er inne i dokumentmappen, så er "`Øving6.pdf`" en relativ filsti (eller filnavn) for dette PDF-dokumentet. Den er relativ fordi den beskriver hvor "`Øving6.pdf`" er i forhold til dokumentmappen. Den er bare rett om vi allerede er inne i dokumentmappen. En filsti som alltid er korrekt kalles en absolutt filsti.

For å beskrive hvor en fil befinner seg uavhengig av hvor vi måtte befinne oss akkurat nå, må vi bruke en absolutt filsti. Den sier alltid hvor en fil befinner seg i forhold til et fast sted (en disk/stasjon i Windows, eller rotkatalogen, /, på Mac).

I Windows ser en absolutt sti typisk ut som

`"C:\Users\bjarne\Documents\Øving6.pdf"`.

I MacOS er den typisk


`"/Users/bjarne/Documents/Øving6.pdf"`.

`"/` kan brukes som separator i alle operativsystemer, men på Windows blir ofte `"\"` brukt istedenfor. Merk at i C++ er `"\"` en spesialkarakter, så hvis du prøver å bruke den direkte i en filsti vil du få feil.

Generelt kan det være lurt å unngå filstier som inneholder spesielle tegn som f.eks. `æøå+`.

Når du skal åpne og lagre filer i koden din så kan du velge om du vil bruke absolutte eller relative stier. Hvis du bare skriver et filnavn, eksempelvis `"testfil.txt"`, så er dette en relativ sti. Programmet ditt vil da forvente at filen befinner seg i samme mappe som programmet selv blir kjørt fra. Filer som du lagrer i programmet vil også havne i samme mappe når du benytter en relativ sti. Ønsker du å åpne eller lagre filer i en annen mappe må du bruke absolutte filstier.

Du kan enkelt legge til filer som skal brukes i prosjektet ved å putte dem i prosjektmappen (altså der `main.cpp` og resten av kodefilene ligger). Prosjektmappe kan på Windows finnes ved å høyreklikke på en fil i prosjektet og velge *Reveal in Explorer*, eller på mac med *Reveal in Finder*.

Nye filer kan i VS Code legges direkte inn ved holde musepekeren over Explorer-vinduet og velge *New File* .

2 Map (10%)

Nyttig å vite: `std::map`

`map` er en assosiativ beholder, dvs. den brukes til å holde styr på to elementer som henger sammen. Hvert element i et `map` er et par, som består av en nøkkel og en verdi. Nøklene må være unike, men verdiene kan være like. I tillegg må det være mulig å ordne (sortere) nøklene, siden elementene i et `map` blir ordnet etter nøklene. For å kunne benytte oss av `std::map` må vi inkludere headeren `<map>`. Et eksempel på et `std::map` er:

```
std::map<string, int> myMap; // her er nøkkelen en string og verdien en int
```

Vi kan legge til elementer i `map`-et med `insert`-funksjonen.

```
myMap.insert({"five", 5});
```

Vi kan aksessere elementer med `at()`.

```
int i2 = myMap.at("five");
```

Merk at selv om `[]` operatoren kan brukes å legge til og lese ut verdier, frarådes det svært å bruke denne. Dette er fordi når verdien som blir etterspurt ikke eksisterer i `map`-et, blir denne verdien satt inn, som senere kan føre til problemer. For eksempel:

```
std::cout << "The value is: " << myMap["nonexistentvalue"] << std::endl;
```

Legger til en verdi med nøkkel "nonexistentvalue", selv om verdien her bare leses ut. Bruk derfor alltid `insert()` og `at()` funksjonene.

Vi kan også legge til elementer når vi deklarerer et `map`.

```
std::map<string,int> example_map{{"two", 2}, {"five", 5}};
```

Vi kan iterere gjennom et `map` med en «for each»-løkke.

```
for(const auto& m:myMap){
    std::cout << "key: " << m.first << " value: " << m.second << ", ";
} // vil gi utskriften: key: five value: 5, key: two value: 2,
```

Du kan lese mer om `std::map` i forelesningsnotater og §21.6 i læreboken.

a) Debuggeroppgave

Koden under forsøker å skrive ut verdiene fra et `std::map`, men feiler ved kompilering.

```
#include "std_lib_facilities.h"

const map<string, string> capitalsMap {
    {"Norway", "Oslo"},
    {"Sweden", "Stockholm"},
    {"Denmark", "Copenhagen"}
};

string getCapital(const string& country) {
    return capitalsMap[country];
}

int main() {
    cout << "Capitals:" << endl;
    for (pair<const string, const string> elem : capitalsMap) {
        cout << getCapital(elem.first) << endl;
    }
    return 0;
}
```

Lim inn koden i VSCode og prøv å kompilere den. Hva forårsaker feilmeldingen som kommer opp? Endre kodelinjen som gir feilmelding, slik at koden fungerer som den skal. Vi forventer følgende utskrift til konsollen:

```
Capitals:
Copenhagen
Oslo
Stockholm
```

Beskriv hva du endret på INGINious.

3 Map: Emnekatalog (35%)

Emner ved NTNU har alltid en emnekode og et emnenavn, for eksempel TDT4102 og Prosedyre- og objekt-orientert programmering. Vi ønsker i denne oppgaven å lage en klasse som kan inneholde en relasjon mellom disse to verdiene slik at vi kan holde styr på alle de forskjellige emnene her på NTNU. Dette kan gjøre ved å benytte et `std::map`. I `CourseCatalog.h` ligger det en deklarasjon av klassen `CourseCatalog` som har en rekke medlemsfunksjoner og én privat medlemsvariabel `courses`. Variablen er av typen `map<string, string>` og skal inneholde emnekode og emnenavn, der emnekoden er nøkkelen og emnenavnet er verdien.

Nå skal du definere de tre funksjonene forklart ovenfor. For å se en oversikt over operasjoner som kan utføres på `map` er kapittel 20.2 et fint sted å starte. Disse operasjonene er definert for alle beholdere i standardbiblioteket, som f.eks. `vector` og `map`. De neste tre oppgavene løses i `CourseCatalog.cpp`. *Hint: `insert()`, `at()` og `erase()` er nyttige funksjoner for denne oppgaven*

a) Definer og implementer funksjonen `addCourse()`.

Funksjonen skal legge til et kurs med emnekode og emnenavn i `courses`. Se funksjonsdeklarasjonen i `CourseCatalog.h` for parameterliste.

b) Definer og implementer funksjonen `removeCourse()`.

Oppgaven løses i `CourseCatalog.cpp` Funksjonen skal fjerne et kurs i `courses` gitt av emnekoden.

c) Definer og implementer funksjonen `getCourse()`.

Oppgaven løses i `CourseCatalog.cpp` Funksjonen skal hente et kurs i `courses` gitt av emnekode.

Nyttig å vite: Overlasting av operatorer og `friend`

Å overlaste en operator er å definere hva en operator, f.eks. `+`, `-`, `<<`, gjør på et objekt av en klasse. En operatoroverlasting er en funksjon der vi bruker nøkkelordet `operator`. Mange operatorer kan enten overlastes som en del av klassen eller utenfor klassen. Når en operator overlastes som en del av klassen må det som står til venstre for operatoren være et objekt av klassen. I denne øvingen skal vi overlaste, `>>` og `<<`, da står det alltid en strøm til venstre for operatoren, ikke et objekt av klassen, slik som vist under. Derfor kan disse bare overlastes utenfor klassen (altså ikke som en del av klasse-deklarasjonen/definisjonen).

```
int a = 10;
std::cout << a;
```

Når vi overlaster en operator utenfor klassen har vi ikke tilgang til de private medlemsvariablene, men noen ganger har vi lyst til å ha tilgang til dem. Dette kan løses ved å bruke nøkkelordet `friend`. Da gir man overlastingen tilgangen til klassens private medlemsvariabler. Når man bruker `friend` så skriver man operator overlasting deklarasjonen i klassen, men den er *ikke* en del av klassen.

Headerfilen (.h/.hpp)

```
class Person{
private:
    int age;
    std::string name;
public:
    Person(int age, std::string name) : age{age}, name{name}{};
    friend std::ostream& operator<<(std::ostream& os, const Person& p);
};
```

Implementasjonsfilen (.cpp)

```
std::ostream& operator<<(std::ostream& os, const Person& p){
    os << "Age: " << p.age << " Name: " << p.name;
    return os;
}
```

Over er et eksempel på overlasting av `<<`-operatoren for `Person`-klassen. Merk at vi ikke skriver `friend` eller `Person::` når vi definerer overlastingen. Når vi har overlattet `<<`-operatoren kan vi både skrive `Person` objekter til fil og ut til konsollen.

```
Person p {20, "Per"};
std::cout << p << std::endl;
// vil gi utskriften: Age: 20 Name: Per
```

Du kan lese mer om overlasting av `<<` operatoren i §8.6 i læreboken og i infobanken til `Cout`

d) Overlast `operator<<`.

Du skal nå bruke `<<`-operatoren til å skrive ut alle emnekodene med tilhørende emnenavn. Funksjonen er allerede deklartert som `friend` av klassen ved følgende deklarasjon i klasse-scopet: `friend ostream& operator<<(ostream& os, const CourseCatalog& c)`. Formatet på utskriften bestemmer du selv, så lenge du bruker HELE emnekode og emnenavnet i utskriften.

e) Test klassen din og sjekk at alt fungerer som forventet.

Implementer testfunksjonen `testCourseCatalog` som ligger i `main.cpp`. Funksjonen skal legge til emnene *TDT4110 Informasjonsteknologi grunnkurs*, *TDT4102 Prosedyre- og objektorientert programmering* og *TMA4100 Matematikk 1* for så å skrive ut en oversikt over emnene. Det er viktig at du bruker nøyaktig disse emnekode (emnenavnene trenger ikke være lik) og skriver det ut til terminalen for at autorettingen skal fungere.

f) Oppdatering av verdier i et map.

Emnet TDT4102 blir som oftest bare kalt for «C++» blant studentene. Oppdater emne-navnet ved å legge til kurset på nytt vha. `addCourse()` i testfunksjonen du lagde i forrige oppgave (uten å fjerne kurset først). Hva skjer?

Det finnes to metoder for å legge til verdier i et `map`: `operator[]` og medlemsfunksjonen `insert()`. Eksperimenter med de ulike metodene i `addCourse()`. Endrer oppførselen til funksjonen seg? Har du en forklaring på hva som skjer? Finnes det en bedre måte å oppdatere verdier i et `map`? Oppgaven besvares på INGINious.

g) Lagre dataene.

Et problem med implementasjonen er at alt som legges inn vil bli slettet hver gang programmet lukkes og må legges til igjen ved oppstart. Implementer derfor medlemsfunksjonen `saveToFile()` som skal lagre alle emnekode og emnenavnene i en tekstfil. Deklarasjonen til funksjonen er i `CourseCatalog`-klassen.

Formatet på dataen skal separere ulike elementer med `:` for å gjøre det lett å lese. Videre skal hvert emne ha vært sin linje i tekstfilen. Et eksempel på en lagret fil er gitt under:

```
TDT4102:Prosedyre- og objektorientert programmering
TMA4100:Matematikk 1
```

h) Last inn dataene.

Nå skal du laste inn dataene fra en fil med samme format som i oppgaven over. Implementer medlemsfunksjonen `loadFromFile()` for å gjøre dette.

4 Animasjon og lesing fra strukturert fil (35%)

I denne oppgaven skal du bruke `AnimationWindow` til å lage en animasjon av en «bouncing ball». Du kan lese om hvordan man lager enkle animasjoner i dokumentasjonen [her](#).

Ballen skal bevege seg fra venstre til høyre og flytte seg opp og ned mellom toppen og bunnen av vinduet. Når ballen kommer til høyre side og «forsvinner» ut av vinduet skal den dukke opp igjen på venstre side, i samme høyde, og fortsette sikksakk bevegelsen. Sikksakk bevegelsen skal ha en vinkel på 1 radian (≈ 60 grader). Ballen skal tegnes som en sirkel med radius 30. Den skal ha en farge når den beveger seg oppover og en annen farge når den beveger seg nedover. Fargene ballen skal ha, og hastigheten til bevegelsen dens, skal leses inn fra den strukturerte filen `konfigurasjon.txt`. Denne filen ligger sammen med den utdelte koden `bouncingBall.cpp/.h` som du skal bruke når du løser denne oppgaven. De utdelte filene kan hentes via TDT4102-extensionen på samme måte som utdelte filer fra tidligere øvinger.

a) Definer structen `Config`. `Config` skal holde tre `public` heltallsverdier som skal tilsvare fargen til ballen på vei opp, fargen til ballen på vei ned, og hastigheten til ballens bevegelse, med variabelnavn `color_up`, `color_down` og `velocity` henholdsvis. Deklarasjonen av `Config` skal ligge i `bouncingBall.h`. Det er viktig at struct-navnet og medlemsvariablene til `Config` er nøyaktig lik som definert i oppgaven.

b) Definer farger. Definer et `map` som kobler hvert av heltallene 1-4 med en farge av typen `Color`. Du kan selv velge hvilke farger. Det vil si at heltallene er nøklene til `map`-et, mens `Color` er verdien. `Map`-et skal hete `ball_colour`.

c) Definer `istream& operator>>(istream& is, Config& cfg)`.

Denne funksjonen skal overlaste `>>`-operatoren. Oppgaven dens er å hente informasjon fra en `istream` og skrive den til vår type `Config`.

Eksempel på filformat of hvordan variablene skal initialiseres:

Vi skal lese denne linjen fra filen `konfigurasjon.txt`

```
1 2 3 3 4 15
```

og lagre den i en `Config`-variabel. `>>`-operatoren skal fungere slik at vi kan gjøre det på følgende måte:

```
std::filesystem::path config_file{"konfigurasjon.txt"};
std::ifstream is{config_file};
Config slow;
is >> slow; // slow.color_up = 1, slow.color_down = 2, slow.velocity = 3
```

MERK: Det er veldig viktig at du følger den samme filformaten som i eksempelet, og at overlastingen av `>>`-operatoren funker selv om det er *flere* linjer i `konfigurasjon.txt`. Dette er viktig for at den siste oppgaven skal fungere som forventet.

Tips: Les om overlasting av `>>`-operatoren i §8.6 i læreboka

d) Lag "bouncing ball" animasjonen som beskrevet i starten av oppgaven.

Filen `konfigurasjon.txt` inneholder to linjer, der hver linje inneholder informasjon om de to ulike fargene og hastigheten som ballen skal ha. Den første linjen gir konfigurasjonen til en treg "bouncing ball" og den andre en raskere en. I funksjonen `bouncingBall()` er det laget en `while`-løkke som kjører så lenge vinduet holdes åpent og skal sørge for tegning av hvert bilde i animasjonen. I denne løkken må posisjonen til ballen endres slik at vi får sikksakk bevegelse, og ballen må tegnes med funksjonen `draw_circle()`. Bevegelsen i x-retning er allerede implementert, inkludert håndtering av når ballen forsvinner ut på høyre side av vinduet. Det er også implementert at ballen bytter farge og hastighet hver 2. gang den "forsvinner" fra høyre side av skjermen.

Det som gjenstår for deg å implementere er:

- Bevegelse i y-retning, enten oppover eller nedover ettersom ballen er på vei opp eller ned.
- Håndtering av når ballen kræsjer i toppen/bunnen av vinduet.
- Tegning av ballen med riktig farge ettersom den beveger seg oppover eller nedover.

Når du skal begynne på oppgaven, må du fjerne

```
#define BOUNCING_BALL
```

fra `bouncingBall.cpp`. Koden ligger rett over implementasjonen av `bouncingBall()`. Dette kalles for en *makro*, og kan brukes sammen med

```
#ifndef BOUNCING_BALL
// Kode
#endif
```

for å fortelle kompilatoren at du *ikke* ønsker å kompilere det innenfor `#ifndef` og `#endif`. Dette er ikke noe du trenger å tenke på når du skal gjøre øvingen, og er kun her for å unngå kompileringsfeil når du får utdelt kode.