Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering
Vår 2026

Øving 6

Frist: 2026-02-22

Mål for denne øvingen:

- Lese fra og skrive til filer
- Strømmer (streams)
- Operatoroverlasting
- Animasjon

Generelle krav:

- Bruk eksakte navn og spesifikasjoner gitt i oppgaven.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal leveres på INGInious.
- 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 standardbibliotektet 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 10 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. I figur 1a og 1b kan du se et eksempel på kjøring av `writeUserInputToFile`. Det er viktig at akkurat ordet `quit` er det som avslutter programmet, ellers fungerer ikke autoretteren.

```
Write some words, they will be output in the file "outputfile.txt",
every word on separate lines. quit exits
Hei!
TDT4102 er så gøy, jeg koser meg!
quit
```

(a) Eksempel på tekst man kan skrive i terminalen.

outputfile.txt

```
1  Hei!
2  TDT4102
3  er
4  så
5  gøy,
6  jeg
7  koser
8  meg!
```

(b) Filen som genereres av `writeUserInputToFile`.

outputfile.txt.linum

```
1  1: Hei!
2  2: TDT4102
3  3: er
4  4: så
5  5: gøy,
6  6: jeg
7  7: koser
8  8: meg!
```

(c) Filen som genereres av `addLineNumbers` i oppgave 1b).

Figur 1: Eksempel på tekst man kan skrive i terminalen og tilsvarende generert fil.

b) Legg til linjenummer.

Implementer funksjonen `addLineNumbers` i `FileUtils.cpp`, 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 ha ekstakt samme navn som den forrige, men med «.linum» på slutten. F.eks. hvis inputfilen er «file.txt», skal outputfilen hete «file.txt.linum». Se figur 1b og 1c for eksempel på kjøring av `addLineNumbers`. 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). Prosjektmappa 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 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 ($\approx$ 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 strukturert 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 en `vector` som inneholder 4 elementer av typen `Color`. Du kan selv velge hvilke farger. Du skal senere bruke heltallene 0, 1, 2 og 3 for å indeksere denne vektoren, dvs. at hvert tall tilsvarer en farge i vector-en. Vektoren skal hete `ball_color`.

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 overlastet `<<`-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 §10.8 i læreboken og i infobanken til *Cout*

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`

0 1 3

2 3 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 = 0, slow.color_down = 1, slow.velocity = 3
```

MERK: Det er veldig viktig at du følger det samme filformatet 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 håndering 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
```

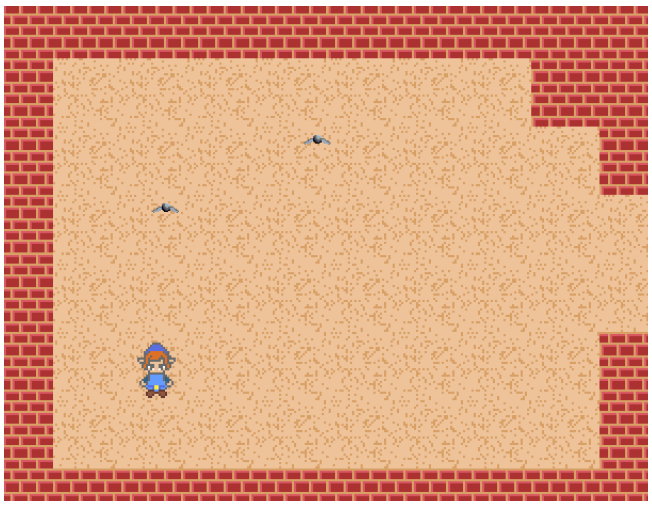
fra bouncingBall.cpp. Koden ligger rett over implementasjonen av `bouncingBall()`. Dette kalles for en *makro*, og kan brukes sammen med

```
#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.

3 Operatoroverlasting og lesing fra strukturert fil (45%)

I denne oppgaven har du fått utdelt kode som inneholder deler av funksjonaliteten til et Zelda-inspirert spill. Du skal implementere resten av funksjonene som trengs for at spillet skal fungere.



Figur 2: Eksempel på hvordan et rom i spillet kan se ut.

Spilleren kontrollerer en avatar som kan bevege seg mellom rom, overvinne fiender og åpne skattekister. Et eksempel på spillvinduet er i figur 2. Utformingen av rommene, antall fiender og fiendenes bevegelsesmønster bestemmes av konfigurasjonsfilene som ligger i mappen `res/maps/`. Hvert rom som kan utforskes i spillet har en egen konfigurasjonsfil, og hver fil består av 4 deler.

- **Størrelse:** Den første linjen i fila angir størrelsen på rommet, med et tall for bredden og ett for høyden, separert med et mellomrom: `"bredde høyde"`.
- **Utforming/layout:** På linjene under står selve utformingen av rommet. Her markeres vegger med `"#"` og åpninger til andre rom med `"E"`.
- **Antall objekter:** På linja under utformingen skal det stå et tall for antall objekter som er i rommet.
- **Objektbeskrivelse:** Hvert objekt i rommet har en linje med beskrivelse, formattert slik: `"Objekttype Objektposisjon"`. Objekttypene tilgjengelig i vårt spill er `Fly` og `Chest`. Et `Chest`-objekt oppgis med én posisjon, mens et `Fly`-objekt objekt oppgis som en bane med punkter den flyr mellom.

I figur 3 kan du se et eksempel på en konfigurasjonsfil. Den fila beskriver et rom med størrelse `10` i bredde og `8` i høyde, som inneholder `2` objekter, begge av typen `Fly`.

Alt av animasjonene til spilleren og fiendene er allerede implementert i den utdelte koden, men den mangler funksjoner for å lese inn konfigurasjonen fra filene i `res/maps/`.

All koden du skal skrive i denne oppgaven skal skrives i `IO.cpp`-fila.

Merk: For å kjøre spillet må du kalle `Dungeon::run()` i `main.cpp`. Dersom du kaller denne funksjonen nå vil spillet kun vise avataren i et tomt, firkantet rom uten noen utganger. Rom med fiender og kister i seg vil ikke genereres før du har fullført alle deloppgavene i denne oppgaven. Det er altså ikke nyttig å kjøre `Dungeon::run()` før du skal teste koden din i oppgave 3d.

a) Definer `istream& operator>> (std::istream& inStream, Dungeon::Map& map)`.
Denne funksjonen skal overlaste `>>`-operatoren for `Dungeon::Map`. Den skal håndtere lesing av de første to bitene av konfigurasjonsfilen som er beskrevet i introduksjonen til denne oppgaven: størrelsen og utformingen til "mappet" eller "rommet". Først skal funksjonen lese bredden og høyden, og deretter bruke medlemsfunksjonen `resize(int width, int height)` til `Map`-objektet for å sette riktig størrelse på kartet.

res > maps > area-0-0.map

```
10 8
##########
#        ##
#         #
#         E
#         E
#         #
#         #
##########
2
Fly 2 2 5 5 5
Fly 2 5 4 2 4
```

Figur 3: En konfigurasjonsfil som beskriver et rom med størrelse `10` i x-retning og `8` i y-retning, som inneholder `2` objekter, begge av typen `Fly`.

Deretter skal funksjonen gå gjennom alle tegnene som beskriver rommet i konfigurasjonsfila, og bruke medlemsfunksjonen `setTileAt(TDT4102::Point location, MapTileType type)` for å sette riktig tile-type på rett posisjon. `MapTileType` er en `enum class` og kan ha verdiene `WALL`, `GROUND` eller `EXIT`. Deklarasjonen til klassen `Map` og `enum`-klassen `MapTileType` ligger i fila `Map.h`. Hvis du er usikker på hvordan medlemsfunksjonene brukes, kan du se gjennom deklarasjonen i `Map.h`. Sammenhengen mellom tegnene i konfigurasjonsfila og `MapTileType` er som følger:

- # - > `MapTileType::WALL`
- ' ' - > `MapTileType::GROUND`
- E - > `MapTileType::EXIT`

NB!: Etter at du har lest bredden og høyden, ligger det et linjeskift igjen i strømmen. Det første kallet til `getline` vil derfor kun lese dette linjeskiftet. Kall `getline` én ekstra gang før du begynner å lese layouten.

b) **Definer `istream& operator>> (std::istream& inStream, Dungeon::Fly& fly)`.** Denne funksjonen skal overlaste `>>`-operatoren for `Fly`. Den skal håndtere lesing av posisjonen til et objekt av typen `Fly`. En linje som beskriver et `Fly`-objekt har formatet: "`Fly n x y x y ...`", der `n` er antall punkter i ruten, og de påfølgende `x`-ene og `y`-ene er koordinater til de `n` punktene. Du kan anta at når denne operatoren brukes så er ordet `Fly` allerede lest fra `istream`, så det første du henter ut fra `istream` i funksjonen er tallet `n`.

Klassen `Fly` har en medlemsvariabel `wayPoints` som er en `vector` av `FloatingPoints`. En `FloatingPoint` er en `struct` som har to variable: `double x` og `double y`. Hvert punkt som leses fra linja om `Fly` skal legges til i `wayPoints`-vektoren.

Til slutt må du bruke medlemsfunksjonen til `Fly`:
`setPatrolRoute(std::vector<FloatingPoint> patrolRouteWaypoints)` for å animere `Fly`-en riktig. Du kan se i fila `Interactables.h` for å finne deklarasjonen til klassen `Fly`.

c) **Definer funksjonen `filesystem::path Dungeon::roomCoordinateToMapFile(TDT4102::Point roomLocation)`.** Denne funksjonen skal ta inn posisjonen til et rom som en `TDT4102::Point` og returnere en filsti til den riktige konfigurasjonsfila for det rommet som en `filesystem::path`. Navnene på konfigurasjonsfilene er formattert slik: `area-x-y.map`. Navnet på konfigurasjonsfila til rommet på posisjon `(0,0)` vil f.eks. være `area-0-0.map`.

Tips: Det finnes en variabel `filesystem::path Dungeon::mapsDirectory` som inneholder filstien til `res/maps`-mappen.

d) Definer funksjonen `void Dungeon::loadRoom(TDT4102::Point currentRoomCoordinate, Dungeon::Map& map, Interactables& interactables)`. Funksjonen skal ta inn en `TDT4102::Point` med posisjonen til et rom, en referanse til et `Map`-objekt og en referanse til et `Interactables`-objetkt, som holder oversikt over alle `Chest`- og `Fly`-objekter i rommet. Den skal bruke `roomCoordinateToMapFile`-funksjonen for å finne filstien til den riktige konfigurasjonsfila, og deretter lese fra fila og sette opp rommet slik som konfigurasjonen tilsier. I denne funksjonen må du benytte overlastingen av `operator>>` til både `Map`, `Fly` og `Chest` for å lese konfigurasjonen fra fila.

For å plassere et `Chest`- eller `Fly`-objekt i rommet må du først opprette en ny instans av `Chest`/`Fly`. Deretter må du bruke overlastingen av `operator>>` for lese inn konfigurasjonen til objektet, og så bruke medlemsfunksjonen `spawn` til klassen `Interactables` for å sette `Chest`/`Fly`-objektet i rommet.

Når funksjonen er korrekt implementert, skal du kunne spille spillet og bevege deg mellom rommene.

Tips: For at rommet skal tegnes riktg, må du også fjerne `Chest`- og `Fly`-objekter fra det forrige rommet. Se `Interactables.h` og `Interactables.cpp` for funksjonalitet som håndterer dette.