



Norges teknisk-naturvitenskapelige  
universitet  
Institutt for datateknologi og  
informatikk

TDT4102 Prosedyre-  
og objektorientert  
programmering  
Vår 2021

Øving 6

**Frist: 2021-02-26**

### Mål for denne øvingen:

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

### Generelle krav:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Teorioppgaver besvares med kommentarer i kildekoden slik at læringsassistenten enkelt finner svaret ved godkjenning.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.
- Det anbefales å benytte en programmeringsomgivelse (IDE) slik som Visual Studio Code.

### Anbefalt lesestoff:

- Kapittel 10, 11, 21.6.1 og B.7 i PPP

## 1 Lese fra og skrive til fil (15%)

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

Definer en funksjon 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. *Hint: du kan lese om hvordan man leser fra fil, og skriver til fil i kapittel 10 i læreboken*

### b) Linjenummer.

Definer en funksjon som leser fra en tekstfil, og lager en ny fil med den samme teksten og har linjenummer som første tegn i hver linje. Sørg for at programmet ditt sjekker for vanlige feil som at filen ikke 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 forelesningsnotater, kap. 11.5 og Appendix B.7.*

#### 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, så 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\Mine Dokumenter\Øving6.pdf"`,


i MacOS er den typisk `"/Users/bjarne/Dokumenter/Øving6.pdf"`.

Merk at i C++ er `\` en spesialkarakter, så hvis du prøver å bruke den direkte i en filsti vil du få feil. Vi løser dette med å enten bruke `\\` som tolkes som en enkelt skråstrek, eller bruke vanlig fremoverskråstrek (`/`) som C++ oversetter til bakoverskråstrek for oss.

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 der når du benytter en relativ sti.

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 Lese fra fil: tegnstatistikk (15%)

I denne deloppgaven skal du lese fra en tekstfil og lage statistikk over bokstavene. For å teste programmet ditt trenger du en tekstfil som inneholder en passende mengde vanlig tekst (minst noen få linjer). Bruk hvilken som helst tekst du vil, eller lag en ny tekstfil med programmet du skrev i del 1.

### a) Tegnstatistikk i en fil.

Programmet skal telle antall bokstaver i filen og hvor mange ganger hver bokstav forekommer. Du kan begrense antall forskjellige bokstaver du teller til normale engelske bokstaver (a-z) og du kan også forenkle oppgaven ved å konvertere alle bokstavene til små bokstaver med `tolower(char)`.

Test funksjonen med den utdelte filen `"grunnlov.txt"`. Filen inneholder Norges grunnlov som ble undertegnet 17. mai 1814. For at du skal kunne verifisere at programmet virker riktig er løsningsforslagets utskrift vist under.

`"grunnlov.txt"` kan hentes via TDT4102-extensionen på samme måte som utdelte filer fra tidligere øvinger.

*Hint: Du kan løse denne oppgaven ved å bruke en **vector** med lengde lik antall forskjellige bokstaver. Hvis du for eksempel tar inn bokstaven 'e', kan du inkrementere elementet i tabellen på posisjon 'e' — 'a' (`characterCount['e'-'a']`). Pass på at du ikke går utenfor grensene til **vector**en.*

*Denne oppgaven kan også løses ved bruk av et **map**, se kapittel 21.6 i læreboken.*

### Statistikk fra grunnlov.txt for tegn a-z:

a: 1923	b: 353	c: 131
d: 1768	e: 4569	f: 709
g: 1355	h: 523	i: 1658
j: 149	k: 538	l: 1433
m: 867	n: 2198	o: 1264
p: 225	q: 2	r: 2230
s: 1863	t: 2145	u: 320
v: 522	w: 13	x: 9
y: 108	z: 6	

### 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 gjøres ved å benytte `map`.

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

Implementasjonsfilen (.cpp)

```
ostream& operator<<(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. Hvis du er usikker på hvorfor spør studentassistenten din.

Du kan lese mer om overlasting av `<<` operatoren i kapittel 10.8 i læreboken.

#### a) Deklarer klassen `CourseCatalog`.

Klassen skal inneholde emnekode og emnenavn i et `map<string, string>`. Medlemsfunksjoner som skal deklarerer:

- `void CourseCatalog::addCourse()` – legg til et kurs med emnekode og emnenavn.
- `void CourseCatalog::removeCourse()` – fjern et kurs gitt av emnekode.
- `string CourseCatalog::getCourse()` – finn emnenavnet til et kurs med en gitt emnekode.

Du må selv avgjøre hvilke parametere funksjonene potensielt skal defineres med, de tomme parentesene er der bare for å indikere at det er funksjoner.

I tillegg skal <<-operatoren defineres for klassen: `friend ostream& operator<<(ostream&, const CourseCatalog&)`, som skal skriv ut alle emnekoder med tilhørende emnenavn.

**b) Definer medlemsfunksjonene.**

For å se en oversikt over operasjoner som kan utføres på `map` er kapittel B.4.7 et fint sted å starte. Disse operasjonene er definert for alle beholdere, som f.eks. `vector` og `map`.

**c) Test klassen.**

Lag en funksjon som legger til emnene *TD4110 Informasjonsteknologi grunnkurs*, *TD4102 Prosedyre- og objektorientert programmering* og *TMA4100 Matematikk 1*. Skriv så ut en oversikt over emnene.

**d) Oppdatering av verdier i et map.**

Emnet TD4102 blir som oftest bare kalt for «C++» blant studentene. Legg til en linje som oppdaterer emnenavnet til TD4102 (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?

**e) 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. Lag derfor en medlemsfunksjon som laster inn informasjonen fra en tekstfil og en medlemsfunksjon som lagrer informasjonen til en tekstfil.

Formatet på dataen du lagrer bestemmer du selv. Du kan f.eks. bruke formatet som du allerede har brukt i den overlastede utskriftsoperatoren. Å separere ulike elementer med spesialtegn gjør det lettere å lese, bl.a. `' '`, `' '` eller `'|'` er ofte brukt.

## 4 Lesing av en strukturert fil (35%)

I denne oppgaven skal filen `"temperatures.txt"` leses. Filen inneholder maksimum- og minimumtemperatur for hvert døgn i perioden 3. februar 2018 - 3. februar 2019.

**a) Definer typen Temps.** Velg en struct eller en klasse ettersom hva du mener er mest hensiktsmessig. `Temps` skal holde to flyttalsverdier: `max` og `min`. Disse verdiene representerer en linje i filen og ett døgn med måledata.

**b) Definer `istream& operator>>(istream& is, Temps& t)`.**

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

**Eksempel:** Vi skal lese en linje fra filen `temperatures.txt`.

```
3.14    -4.0
```

For å lese den inn i programmet vårt skal operatoren fungere som følger:

```
ifstream temp_file{"temperatures.txt"};
Temps t;
temp_file >> t; // t.max = 3.14, t.min = -4.0
```

**c) Definer funksjonen `readTemps()`**

Funksjonen skal ta inn et filnavn som parameter og returnere en `vector` med alle temperaturene i filen;

**d) Definer funksjonen tempStats()**

Funksjonen skal ta inn en **vector** med temperaturer og skrive ut statistikk for temperatu-  
rene til skjerm. Statistikk som skal være med er:

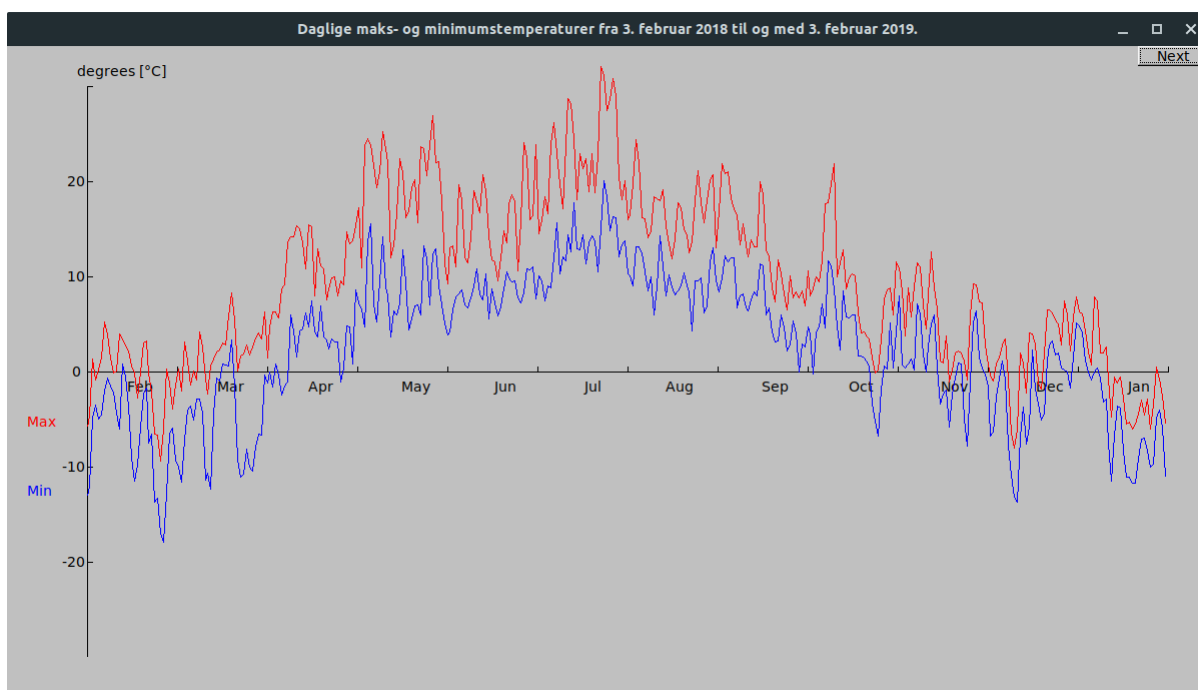
- Hvilken dag som hadde høyest temperatur og hva denne temperaturen var.
- Hvilken dag som hadde lavest temperatur og hva denne temperaturen var.

Du trenger kun finne ut hvilken indeks dagene er på, ikke hvilken dato de er.

**5 Plotting av graf — frivillig —**

Visualiser temperaturdataene du har lest fra filen med grafikk.

Løsningsforslaget til denne oppgaven er nesten helt lik fremgangsmåten som er gjennomgått i kapittel 15.6 i PPP og produserer grafikken som vises i figuren under, se gjerne på dette kapittelet før du gjør oppgavene.

**a) Definer symbolske konstanter for vinduet.**

For å enkelt kunne skalere grafer og akser slik at det ser pent ut definerer vi en rekke konstanter. Her lønner det seg å tegne opp hvordan du ønsker at vinduet skal se ut, og definere konstanter ut fra dette, se på figuren i §15.6.2 i læreboken.

**Nyttig å vite: Axis**

**Axis** er en klasse som gjør det enkelt å sette opp aksene til et koordinatsystem. En **Axis** opprettes slik:

```
Axis(Orientation d, Point xy, int length, int n, string lab)
```

Man oppretter ett **Axis**-objekt per akse, **d** er retningen du ønsker aksene i, **x** eller **y**. **xy** er punktet aksene starter i og **length** er aksens lengde. **n** er antall målemerker langs aksene. **lab** er aksens tittel, dette er et **Text**-objekt som kan flyttes vha. **Axis::label.move(int x, int y)**. Du kan se eksempler på bruk av **Axis** i §15.6.4 i læreboken.

**b) Lag et vindu med akser.**

Lag et **Simple-window**-objekt, x- og y-aksen kan lages som **Axis**-objekter, bruk konstantene fra forrige deloppgave til å plassere aksene i vinduet. Langs x-aksen skal det være målemerker for hver måned. Målemerkene på x-aksen kan navngis ved å inkludere månedene i tittelen og deretter flytte det slik at det passer med merkene. Y-aksens målemerker kan beskrives vha. separate **Text**-objekter.

**c) Skalering.**

For at grafene skal passe til koordinatsystemet må punktene skaleres. Finn konstanter **xscale** og **yscale** basert på forholdet mellom data-spredningen og aksenes lengde. Skaleringsfaktoren tar ikke hensyn til offset i koordinatsystemet så det trengs flere kalkulasjoner for å plassere en verdi langs aksene. Dersom man ser på en lengderetning i koordinatsystemet kan man finne plasseringen fra formelen:

$$\text{plassering} = \text{koordinatoffset} + (\text{verdi} - \text{verdioffset}) * \text{skaleringsfaktor}$$

**Eksempel:** Vi har et koordinatsystem med en x-akse som har offset = 100 fra kanten av vinduet (*koordinatoffset*), skaleringsfaktor **xscale** = 0,01 og x-verdien i origo er **x** = 5000 (*verdioffset*). Punktet (600, 50) skal dermed plasseres langs x-aksen, ved

$$x = 100 + (6000 - 5000) * 0,01 = 200$$

Et smart triks for å unngå mye regning er å lage en egen klasse som kan finne plasseringen. Lag klassen **Scale** som kan regne ut plasseringer basert på skalering og offset, denne må inneholde tre medlemsvariabler. Den trenger to heltallsvariabler, **cbase** som representerer koordinatoffset og **vbase** for verdioffset, i tillegg bør klassen inneholde skaleringsfaktoren. **Scale** må også ha en medlemsfunksjon, **calculatePoint(int value)** som tar inn en verdi og returnerer den korrekte plasseringen basert på formelen over. Opprett to objekter av **Scale**-klassen, et for x- og et for y-aksen.

**Eksempel:**

Hvis vi ønsker å plassere punktet fra i sted (600, 50), kan vi bruke funksjonene og plassere det i punktet

```
(x.calculatePoint(600), y.calculatePoint(50))
```

**d) Tegn grafene.**

Opprett **Open-Polyline**-objekter for grafene **min** og **max**. Iterer gjennom vektoren med **Temps**-objekter og skaler verdiene i både x- og y-retning vha. medlemsfunksjonen til **Scale**. For hvert **Temps**-objekt, kan man lage to **Point**, ett for **max** og ett for **min**, som kan legges til på hver sin graf. Gi grafene ulik farge og navn, du kan bruke **Text**-objekter til dette. Husk å feste alle objektene til vinduet!