



Norges teknisk-naturvitenskapelige
universitet
Institutt for datateknologi og
informatikk

TDT4102 Prosedyre-
og objektorientert
programmering
Vår 2019

Øving 6

Frist: 2019-02-22

Mål for denne øvingen:

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

Generelle krav og anbefalinger:

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

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.

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"`.

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.

I Visual Studio kan filene du bruker i programmet legges til i prosjektet, ved å legge filen til under mappen "Resource Files" i Solution Explorer. Nye filer kan legges direkte inn ved å høyreklikke og velge *Add->New Item*, men eksisterende filer bør legges inn i prosjektmappa før du velger *Add->Existing Item*.

I XCode kan du finne ut hvor dette er ved å velge «Show in Finder» fra kontekstmenyen til programmet i Products/Produkter i Prosjektnavigatoren. Hvis du ikke finner filene programmet ditt har skrevet til, kan du finne informasjon om hvor filene blir av [her](#).

Hvis du ønsker at filene skal havne et spesielt sted så kan du bruke en absolutt sti. Da må du først finne ut hvor stedet du ønsker at filene skal være er, og benytte denne absolutte stien når du skal åpne filene.

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.

*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.*

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: Emne katalog (30%)

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`, en datastruktur i standardbiblioteket til C++ som implementerer en assosiativ, eller innholdsadressert, tabell. Den lar deg lage relasjoner mellom ulike elementer, for eksempel `"hei" → 3` slik at om du slår opp på `myMap["hei"]` får du 3 i retur. Du kan lese om `map` i forelesningsnotater, i kapittel 21.6 og B.4.

a) Deklarer klassen `CourseCatalog`.

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

- `friend ostream& operator<<(ostream&, const CourseCatalog&)` – skriv ut alle emnekode med tilhørende emnenavn.
- `CourseCatalog::addCourse()` – legg til et kurs med emnekode og emnenavn.
- `CourseCatalog::removeCourse()` – fjern et kurs gitt av emnekode.
- `CourseCatalog::getCourse()` – finn emnenavnet til et kurs med en gitt emnekode.

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

Nyttig å vite: friend

Nøkkelordet **friend** gjør det mulig for andre funksjoner og klasser å få direkte tilgang til private medlemmer av klassen.

Merk at **friend** kun kan brukes i en klassesdeklarasjon. Hvis en funksjon skal være **friend** av en klasse må det være en deklarasjon av funksjonen i klassesdeklarasjonen med ordet **friend** foran. Se deloppgave a) for hvordan det ser ut for utskriftsoperatoren i denne oppgaven.

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 (20%)

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.

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) **Les Temps fra fil.** Les alle temperaturene fra filen og lagre dem som Temps-objekter i en vector.

5 Lesing av temperaturdata og plotting av graf (20%)

Visualiser temperaturdataene du har lest fra filen med enkel grafikk. Bygg på eksempler fra forelesning og ved å følge fremgangsmåten i kap. 15.6 i PPP.

Løsningsforslaget til denne oppgaven er nesten helt lik fremgangsmåten som er gjennomgått i kapittel 15.6 og produserer grafikken som vises i figuren under.

Følgende objekter er brukt:

- Grafene er `Open_polyline`-objekter, et objekt for hver graf.
- Det er to `Axis`-objekter (x- og y-aksen).
- Det er fem separate `Text`-objekter som beskriver målemerkene (-20, -10, 0, 10, 20) på y-aksen.
- To `Text`-objekter for å tegne rød og blå tekst (Max og Min).

