

TDT4102

Prosedyre- og objektorientert programmering, Øvingsforelesning veke 10, 8. mars

The image features a large, stylized blue 'C++' logo. Behind the logo is a snippet of C++ code in a monospaced font, tilted at an angle. The code is:

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello World!\n";
    return 0;
}
```

Sivert Krøvel
sivertkr@stud.ntnu.no

Agenda

- Straumar (streams)
- Filbehandling
- Tekstbehandling
- std::map
- Oversikt, øving 8
- Kort om øving 9

Straumar (streams)

- Kopling mellom programmet vårt og «utsida»
- Ein måte å lese/skrive data
- Ein *istream* sender data *inn* til programmet (input)
- Ein *ostream* sender data *ut* (output)

Streams

Vi har allerede brukt straumar ganske mykje:

`std::cout` og `std::cin`

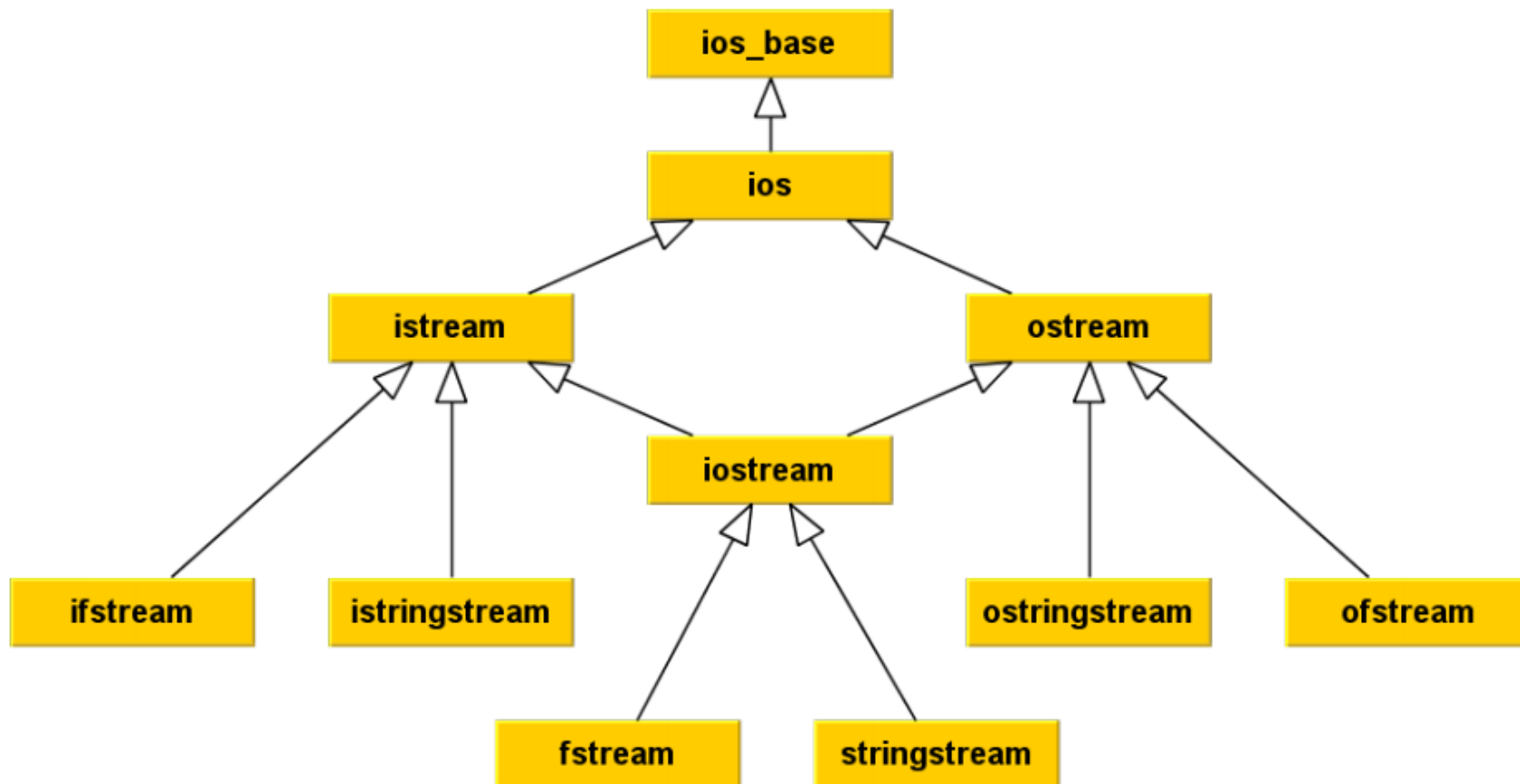
Dei andre straumane fungerer på same måte

Streams

I denne øvinga skal vi bruke *ifstream* og *ofstream* til å overføre data mellom filer og programmet vårt

Vi brukar også *stringstream*, som fungerer ganske likt

Forskjellige streams



Typiske funksjonar

Inputstream:

get, getline, operator>>

Outputstream:

put, write, operator<<

Agenda

- Straumar (streams)
- Filbehandling
- Tekstbehandling
- std::map
- Oversikt, øving 8
- Kort om øving 9

Filbehandling

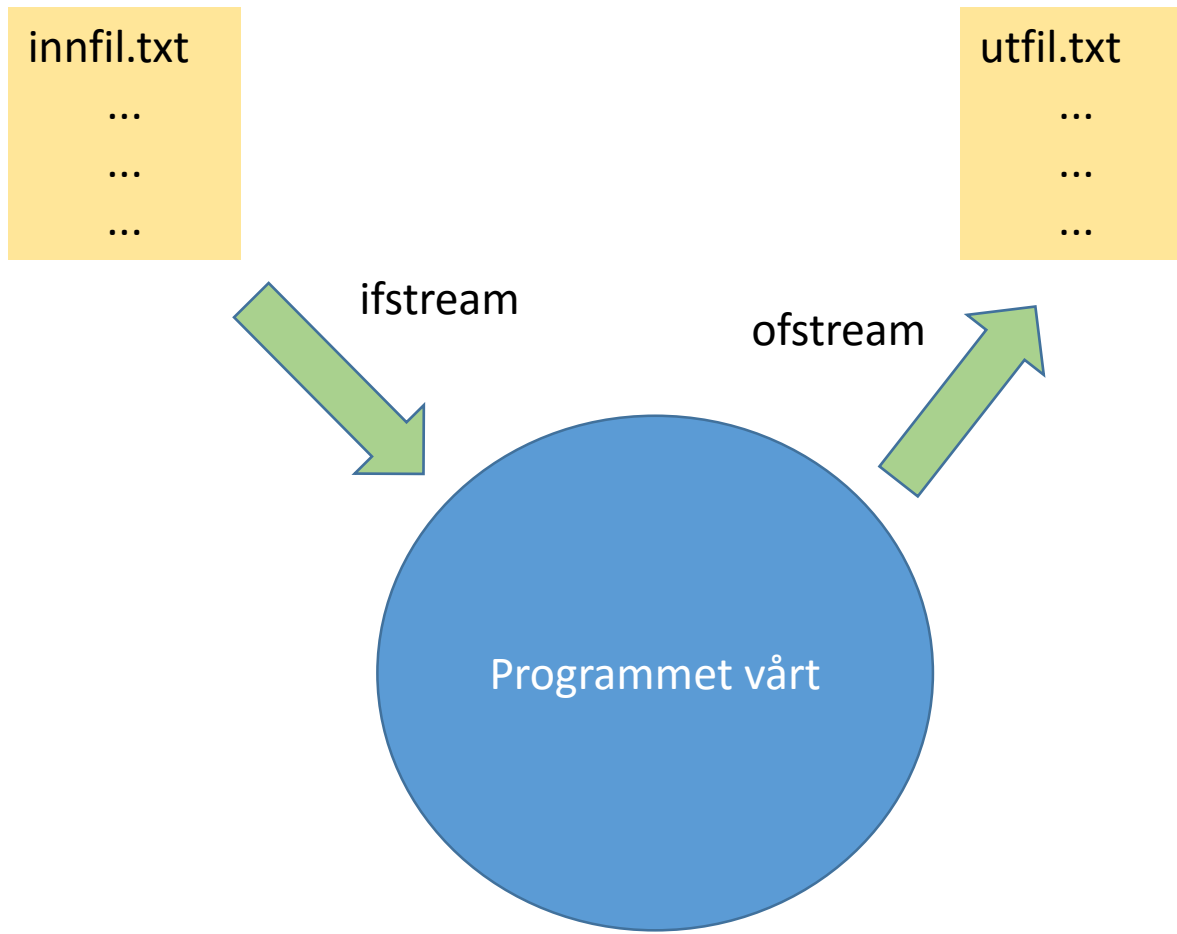
Filer ligg lagra på harddisken, og kan ta vare på resultat frå programmet sjølv etter det er ferdig

I øvinga denne veka lærer vi korleis vi kan lese og skrive til (tekst)filer

Filbehandling

Vi koplar saman filer og programmet vårt ved hjelp av *straumar* (streams)

Illustrasjon, straumar og filer



Straumar koplar programmet til «utsida»

Framgangsmåte

- 1) Opprett ein *ifstream/ofstream*, og åpne fila (kople saman streamen og fila)
- *open()*
- 2) Sjekk at fila blei åpna
- *fail()*
- 3) Utfør lesing/skriving
- *get()*, *getline()*, *<<*, *>>* osv
- 4) Lukk fila
- *close()*

Døme, skrive til fil

Åpne fila

Sjekk at det
gjekk

Utfør
lesing/skriving

Lukk fila

```
ofstream myFile;
myFile.open("testfil.txt");

if (myFile.fail()){
    cout << «File could not open\n";
    return;
}

for (int i = 0; i < 10; i++){
    myFile << "This is line number " << i << endl;
}

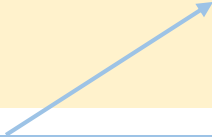
myFile.close();
```

Skrive til eksisterende filer

I programmet frå forrige slide vert eventuelt innhald i fila fjerna, og vi startar å skrive til ei tom fil.

Vi kan legge til ekstra argument i kallet til *open()* for å spesifiere korleis vi vil skrive. Vi kan til dømes velge å legge til på slutten av fila:

```
ofstream myFile;  
myFile.open("testfil.txt", ios::app);  
...
```



Vi legg til *ios::app*, for å signalisere at vi vil legge til innhald på slutten av fila (append). Dersom fila er tom eller ikkje eksisterer, skriv vi frå starten av fila.

Filnamn?

Ei fil har mange namn. Det eine er filnavnet (t.d. "testfil.txt"). Dette brukar vi til å åpne fila. I programmet referer vi til fila med ein *ifstream/ofstream*

Filnamn?

Vi bruker det «ekte» filnamnet kun når vi åpner fila

```
ofstream myFile;
myFile.open("testfil.txt");
...
myFile << "This is the first line\n";
myFile.close();
```

Deretter bruker vi i/ofstream-objektet til å gjøre ting med fila

Lese frå fil (1)

Den tradisjonelle måten (les eitt teikn om gongen):

```
char tegn;  
myFile.get(tegn);  
//get(char& c) lagrar neste teikn i streamen  
//i inputvariabelen  
  
while( ! myFile.eof()){  
    //eof() sjekkar om vi har lest utanfor fila  
    cout << tegn;  
    myFile.get(tegn);  
}
```

Lese frå fil (2)

Les eitt teikn om gongen, alternativ 2

```
char tegn;  
//opprettar ein char til å lagre teiknet  
  
while(myFile >> tegn){  
    //returverdien frå >> kan tolkast som  
    //ein bool, sant dersom vi klarte å lese,  
    //usant elles  
    cout << tegn;  
}
```

Tips: dersom du brukar ein string-variabel i staden for char, les denne metoden inn fila *ord for ord*

Lese frå fil (3)

Les eitt ord om gongen

```
string ord;  
//opprettar ein string til å lagre ordet  
  
while(myFile >> ord){  
    //returverdien frå >> kan tolkast som  
    //ein bool, sant dersom vi klarte å lese,  
    //usant elles. Skiller ord på whitespace  
    cout << tegn;  
}
```

Lese frå fil (4)

Lese inn linje for linje (ved hjelp av getline)

```
string linje;  
//opprettar ein string til å lagre linja  
  
while(getline(myFile, linje)){  
    //returverdien frå getline kan tolkast som  
    //ein bool, sant dersom vi klarte å lese,  
    //usant elles  
    cout << linje;  
}
```

Framgangsmåte (vi repeterer)

- 1) Opprett ein *ifstream/ofstream*, og åpne fila (kople saman streamen og fila)
- *open()*
- 2) Sjekk at fila blei åpna
- *fail()*
- 3) Utfør lesing/skriving
- *get()*, *getline()*, *<<*, *>>* osv
- 4) Lukk fila
- *close()*

Demonstrasjon

Hamlet

Agenda

- Straumar (streams)
- Filbehandling
- Tekstbehandling
- std::map
- Oversikt, øving 8
- Kort om øving 9

Litt om tekstbehandling

I denne øvinga får vi også bruk for litt enkel tekstbehandling. Det gjer seg hovudsaklig gjeldande når vi skal «renske» ein tekststreng for uønska teikn (til dømes komma og punktum)

Nokre nyttige funksjonar

- **isalpha**: sjekkar om argumentet er ein bokstav
- **islower/isupper**: sjekkar om argumentet er lower/upper-case
- **tolower**: returnerer ein lower-case variant av argumentet (dersom det er ein bokstav)
- **toupper**: omtrent det same som over

Demonstrasjon

```
void cleanString(string& str)
```

Agenda

- Straumar (streams)
- Filbehandling
- Tekstbehandling
- std::map
- Oversikt, øving 8
- Kort om øving 9

Standard Template Library

STL er ei samling med praktiske klasser i standardbiblioteket. Vi har allerede brukt *containerklassa* `std::vector`.

I denne øvinga skal vi bruke ei ny containerklasse, nemlig `std::map`

Containerklasse: ei klasse som har som formål å oppbevare (contain) andre objekt

std::map

Denne klassa implementerer det vi på norsk kan kalle ein *assosiativ tabell* (eng: associative array) eller *innhaldsadressert tabell*.

Det går også an å tenke på *map* som ein tabell der indeksen ikkje treng å vere eit heiltal

std::map

- Knyttar saman to verdier (like eller ulike datatypar)
- Den eine datatypen fungerer som *nøkkel* (key) og den andre som *verdi* (value).
- Treng to datatypar i deklarasjonen:

```
map<string, int> myMap;
```

keytype

valuetype

std::map

- Elementa i *map* består av eit *std::pair<keytype, valuetype>*
- Elementa er sorterte på *nøkkelen*
- Kan ikkje innehalde duplikat av *nøkkelen*. Fleire *nøklar* kan derimot ha same verdi

Legge til element (1)

En måte er å bruke indeksoperatoren. Dersom nøkkelen ikke allerede fins i *mapet*, blir den lagt til:

```
map<string, int> myMap;  
//keytype er string, valuetype er int  
  
myMap["hei"] = 4;  
//Legg inn talet 4 på plassen til "hei"
```


Legge til element (2)

Vi kan også bruke medlemsfunksjonen *insert*. Denne tek inn ein instans av typen *std::pair<keytype, valuetype>*:

```
map<string, int> myMap;  
//keytype er string, valuetype er int  
  
myMap.insert(pair<string, int>("hei", 4));  
//Legg inn talet 4 på plassen "hei"
```

Fjerne element

Eit element kan fjernast frå eit *map* ved hjelp av medlemsfunksjonen *erase()*. Den tek inn ein *key* som skal slettast. Dersom denne ikkje fins i *mapet*, skjer ingenting

```
...  
myMap.erase("hei");  
//Fjernar elementet med nøkkelen "hei",  
//dersom det eksisterer
```

Iterere gjennom elementa

Vi brukar iteratorar for å iterere gjennom eit *map*. Iteratoren kan behandlast som ein *peikar* til elementet (eit *std::pair*). Vi finn *nøkkelen* i medlemsvariabelen *first*, og *verdien* i *second*

```
auto it = myMap.begin();
while (it != myMap.end()){
    cout << it->first << ": « << it->second
        << endl;
    ++it;
}
```

Agenda

- Straumar (streams)
- Filbehandling
- Tekstbehandling
- std::map
- Oversikt, øving 8
- Kort om øving 9

Øving 8

Tema:

- Fil input/output
- Map-datastrukturen

Fil I/O

I dei første oppgåvene trenar vi på filbehandling. Framgangsmåten er som skildra tidligare (slide 10). Tenk over før du startar om du bør lese inn filene teikn for teikn eller linje for linje

Emnekatalog (oppg 3)

Vi skal lage ei klasse *CourseCatalog*

- Lagrar ei oversikt over emne, med emnekode og namn
- Medlemsfunksjonar for å legge til og fjerne emne
- Overlastar *ostream* <<-operatoren

Emne katalog

Brukar eit *std::map* til å lagre emnenamn og emnekode

Skal bruke emnekode som «nøkkel»

Ordstatistikk (oppg 4)

Vi skal lage ei oversikt over innhaldet i ei fil. Denne skal innehalde iallefall:

- antal ord
- antal forekomstar av eit ord
- det lengste ordet
- antal linjer

Ordstatistikk

For å lagre ordforekomst, vil vi gjerne ha ein «tabell» med ein *string* som indeks
=> `std::map<string, int>` er ein god idé

Ordstatistikk, tips

1) Det er lurt å lese inn fila linje for linje. Vi får då ei løkke som køyrer éin gong for kvar linje i fila (dreg vi nytte av det?)

2) For kvar linje bør vi lese inn eitt og eitt ord. Her er det nyttig å bruke *stringstream*. Denne løkka køyrer éin gong for kvart ord i fila

Ordstatistikk, tips

3) Når vi har eit ord, bør vi «renske» det før vi legg det inn i statistikken. Slik unngår vi mellom anna å skilje mellom liten og stor forbokstav («Hei» og «hei» bør teljast som det same ordet)

4) Ta alltid vare på det lengste ordet du har funne!

Agenda

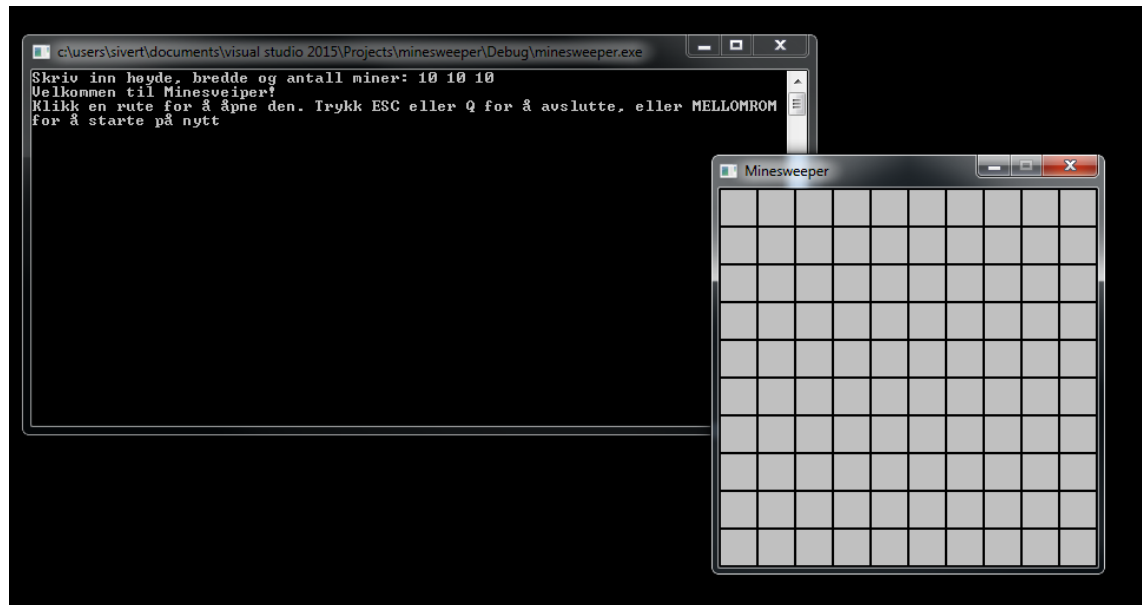
- Straumar (streams)
- Filbehandling
- Tekstbehandling
- std::map
- Oversikt, øving 8
- Kort om øving 9

Kort om øving 9

- Vi skal implementere klassikaren *Minesveiper*.
- Vi brukar SFML også i øving 9, men (nesten) all koden relatert til SFML er denne gongen utdelt.
- Windowsbrukarar: Last ned SFML-prosjektet på nytt, og legg til dei utdelte filene slik som i øving 6

Kort om øving 9

Den utdelte koden skal gi dette resultatet (etter å ha spurt om dimensjonar):



No gjenstår berre å implementere Minesweeper-klassa.