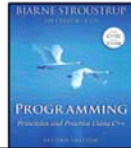
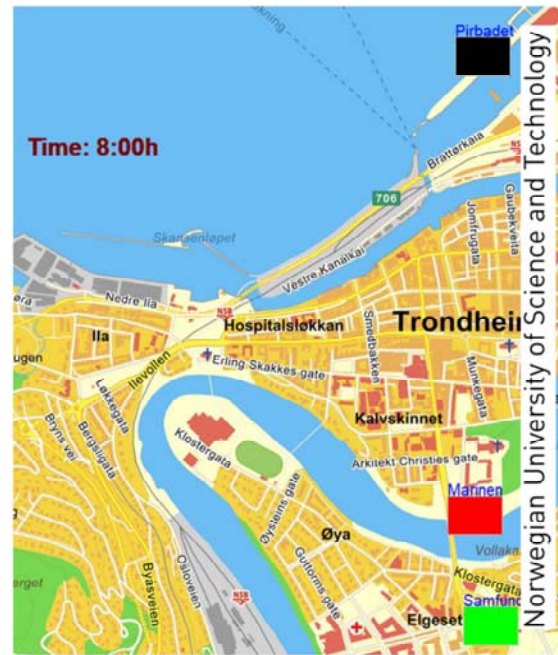


TDT4102 - Procedural and Object-Oriented Programming – Lecture 7

Operator overloading examples, `<map>`, input/output, files, formatting, `cleanAir_V1.cpp`



Some of the slides are adapted from slides by Bjarne Stroustrup found from www.stroustrup.com/Programming



Overview Lecture 7

- Repetition
 - Overloading operator<<
 - Larger example `cleanAir_V1.cpp` (later in the lecture)
- Associative arrays `<map>`
- `static` member variables
- `istream`, `ostream`, files
- User defined input operator
- Example `cleanAir_V1.cpp` (first version)
- More on output, and file modes



User-defined output: `operator<<`



Repetition

```
3 struct Date {
4     int day, month, year;
5 };
6 ostream& operator<< (ostream& os, const Date& d) {
7     return os << '(' << d.year
8         << ',' << d.month
9         << ',' << d.day << ')';
10 }
```

- If `d1` is a `Date`-object, the meaning of `cout << d1;` is the call `cout = operator<<(cout,d1);`

3

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

☐ Brukerdefinert utskriftsoperator `<<`

- Et annet eksempel ble vist i forelesning 6 for enum-klassen `Month`
- Her er samme teknikk vist for en veldig enkel klasse `Date`. Det er det samme som i PPP kap. 10.8, men der er det en litt annen klasse `Date` som er benyttet
- I denne forelesningen, nr. 7, skal dere også lære om `ostream` (output stream)
- vi skriver ikke `operator<<(cout... osv.)` i koden, men det er med på lysarket for å forklare hva som skjer
- Definisjonen av `operator<<` tar inn en `ostream`-referanse som argument og returnerer en `ostream`-referanse. Dette er for å gjøre det mulig å kjede sammen flere gangers bruk av `<<` i en setning, som vi allerede har sett brukt mange ganger. (Se også notater for slide 25 i forelesning nr. 6)

Introduction to `<map>`

- An «associative array» (look-up by content (content-addressable))
- In contrast to vector – subscript using an integer
- **map** – define the “subscript” to be (just about) any type


```
1 // map.cpp
2 #include "std_lib_facilities.h"
3 int main(){
4     map<string, int> words; // keep (word,frequency) pairs
5     for (string s; cin >> s; )
6         ++words[s]; // note: words is subscripted by a string
7                     // words[s] returns an int&
8                     // the int values are initialized to 0
9     for (const auto& p : words)
10         cout << p.first << ": " << p.second << "\n";
11     return 0;
12 }
```

Key type

Value type

first gives access to key type

second gives access to value type



Microsoft Visual Studio Debug Console

```
aaa bbb ccc bbb ccc ccc
^Z
aaa: 1
bbb: 2
ccc: 3
```

❑ Assosiativ tabell, innholdsadressert tabell

❑ Map er forskjellig fra vector som krever at indeks er et heltall

❑ Map er svært nyttig

- Kan ta litt tid å venne seg til å benytte den (Å komme på at den kan brukes)
 - Derfor introduserer vi den tidlig i forelesning
 - Finnes i python

❑ Progameksempel fra læreboka som teller antall forekomster av ord

❑ Forklaring av kodelinjer

- (4) deklarer ett «map» (en avbildning) fra string til heltall, som kalles words
- (5) løkke som leser en og en string fra cin og lagrer i s, terminerer når det ikke er flere string å lese (Ctrl-Z er end of- input/file på windows)
- (6) Hvis s ikke er funnet før vil det bli opprettet ett nytt element (innslag) i avbildningen words, og heltallsverdien økes med 1 (fra initiell verdi 0). Har man funnet den strengen før, så vil man ikke opprette ett nytt element (innslag) men inkrementere tallverdien. Slik teller man opp antall forekomster av s.
- (9) for løkke som bruker auto og løper igjennom alle elementer i words, const garanterer at man ikke endrer noe i tabellen.
- (10) medlemsvariabelen first (som alle map har) gir tilgang til nøkkelen. Tilsvarende gir second tilgang til verdien.

<map> access by [] or .at()

cppreference.com



Element access

<code>at</code> (C++11)	access specified element with bounds checking (public member function)
<code>operator[]</code>	access or insert specified element (public member function)

<https://en.cppreference.com/w/cpp/container/map>

5

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

- ❑ Medlemsfunksjonen `at()` kaster et unntak av type `std::out_of_range` hvis elementet vi prøver å aksessere ikke finnes, mens derimot medlemsoperatoren `[]` vil sette inn et element dersom det ikke finnes fra før
- ❑ Hvis et map er deklartert som `const`, så kan du ikke bruke `[]` på det

<map> example from Exercise 5

```
□ const map<Suit, string> suitStrings {  
    {Suit::clubs, "Clubs"},  
    {Suit::diamonds, "Diamonds"},  
    {Suit::hearts, "Hearts"},  
    {Suit::spades, "Spades"}};
```

- `Suit` is a scoped enum
- `suitStrings` maps from an enum value to a string
- A similar `suitStringsNO` could be used to map to the Norwegian "Kløver", "Ruter", "Hjerter" and "Spar"



static member variable, example

- A **static** member variable exists as one shared instance, instead of one per object of the class

Declares member variable to be static

```
45 class Year { // year in [min:max) range
46     static constexpr int min = 1800;
47     static constexpr int max = 2200;
48 public:
49     class Invalid { };
50     Year(int x) : y{ x } {
51         if (x < min || max <= x) {
52             throw Invalid{};
53         }
54     }
55     int year() { return y; }
56 private:
57     int y;
58 };
```

Declares an empty class named Invalid used as exception

A non-const static member variable must be declared and initialized outside the class (eg. as `int ClassName::objectID = 0;`) --- this will be demonstrated in [cleanAir.cpp](#)



7

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

❑ Eksempel fra læreboka side 325

❑ Man tenker seg at programmet bare håndterer årstall mellom år 1800 og 2200.

- Konstruktøren Year(int x) sjekker om argument x er gyldig, og hvis ikke kastes et unntak av type Invalid, som er en tom klasse deklart lokalt bare her. Dette er en teknikk som brukes en del for å gi en bestemt situasjon et eget navn.
- Konstantene min og max representerer det tallområdet klassen er ment å skulle virke for
- Når de deklarerer som static vil det bare bli lagret én felles kopi av den, i stedet for en kopi av konstanten inne i hvert objekt

❑ En static medlemsvariabel som IKKE er konstant blir som en felles variabel for alle objekter av den klassen, den må initialiseres utenfor klassedefinisjonen, og det ser ganske likt ut som en global variabel

- Men den kan bare brukes for objekter av denne klassen
- Meget nyttig for noen spesielle behov, se eks. i `cleanAir.cpp`
- Generelt bør man IKKE bruke globale variable

❑ Skal vi fange dette unntaket i hovedprogrammet så kan vi gjøre det på to måter:

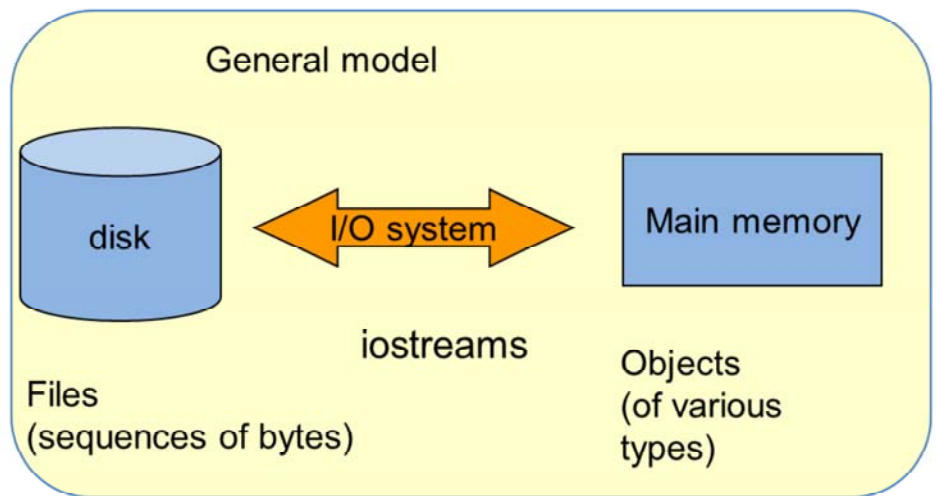
```
catch (Year::Invalid& e) {
    return 2;
}
catch (...) {
    return 1;
}
```

TDT4102: --- Øving 5 – skal være «dekket» ---

Input and output streams



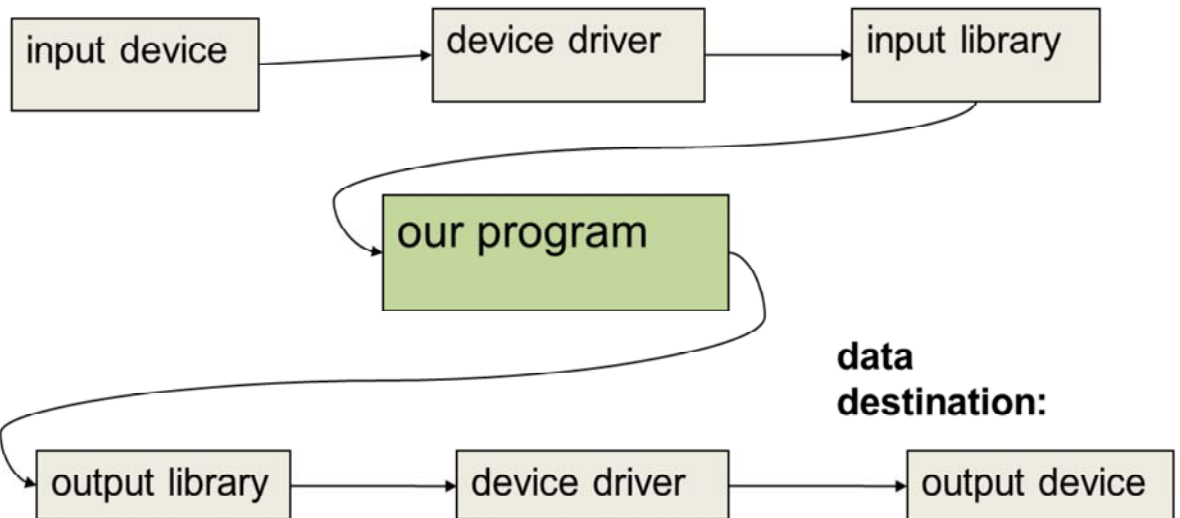
- Fundamental I/O concepts
- Files
 - Opening
 - Reading and writing streams
- I/O errors



☐ Mye av dette overlapper med IT-GK

Input and Output

data source:



10

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

☐ Inndata og utdata

☐ Figuren viser situasjon/omgivelsene til vårt program

☐ Data source = datakilde

☐ Data destination = datamottaker

☐ Input device = inntput-enhet, inndata-enhet

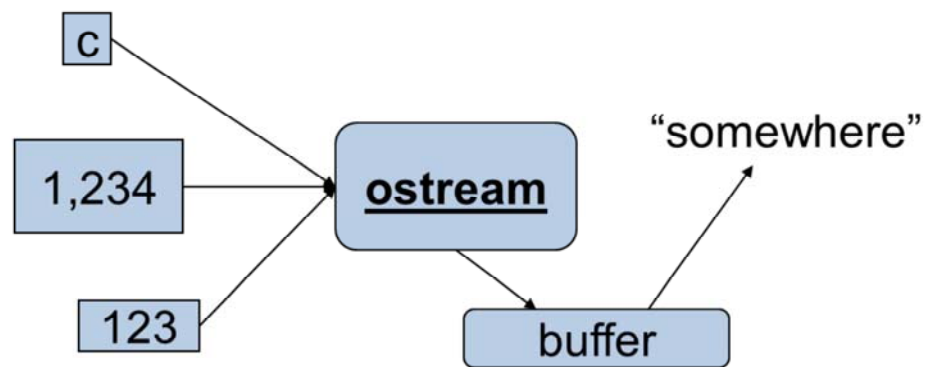
☐ Device driver = enhetsdriver (?)

- dette har vi hatt siden «datamaskinenens morgen» og jeg har aldri hørt noen si inndata-enhet-driver, folk sier «device-driver» på norsk. Men en del bruker terminologi som tastatur-driver, harddisk-driver, fil-system-driver, printer-driver, mus-driver osv.

☐ Input og output bibliotek

- Det som vårt C++ program benytter

The stream model, **ostream**

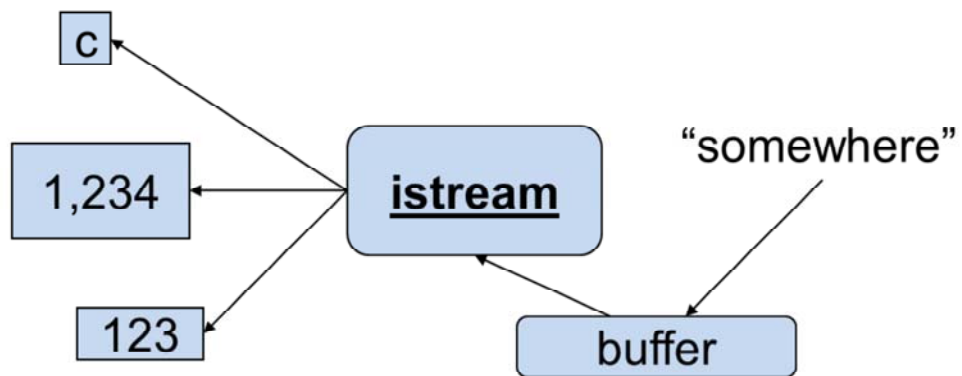


▪ An **ostream**

- turns values of various types into character sequences
- sends those characters somewhere
 - *E.g.*, console, file, main memory, another computer

□ **Ostream** = output stream, ut-strøm

The stream model, **istream**



▪ An **istream**

- turns character sequences into values of various types
- gets those characters from somewhere
 - *E.g.*, console, file, main memory, another computer

□ **Istream** = input stream, inn-strøm

cin and cout, files

- cin is a standard istream
 - >> is the input operator
- cout is a standard ostream
 - << is the output operator
- You can define << and >> for your own types
- A file is a sequence of bytes numbered from 0 upwards
- Text files and binary files

0: 1: 2:



13

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

- ☐ Hvordan vi kan definere << eller >> på våre egne datatyper vil vi demonstrere i eksemplet cleanAir.cpp. Dette kalles operatoroverlasting
- ☐ Vi skiller ofte mellom tekst-filer og binær-filer
 - Selv en tekstfil kan tolkes etter et bestemt format.
F.eks. « 123,4 5,678» kan programmet tolke som to flyttall adskilt med ett eller flere blanke tegn, eller det kan være en ren tekstfil som en sekvens av tegn, inkludert de blanke
- ☐ cin, cout, cerr og mye filhåndtering er tekst-filer
- ☐ Binær-filer håndteres på en litt annen måte
 - Dekket i kapittel 11

Opening a file for reading

An ifstream is an istream for reading from a file

```
1  #include "std_lib_facilities.h"
2  int main() {
3      cout << "Please enter input file name: ";
4      string iname;
5      cin >> iname;
6      ifstream ist{ iname }; // ifstream is an "input stream from a file"
7                             // defining an ifstream with a name iname
8                             // opens the file of that name for reading
9      if (!ist) error("can't open input file ", iname);
10 }
```

14

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

□ Kodelinjer

- (3-5) Ber bruker taste inn navn på filen som skal åpnes
- (6) definerer ist som å være av type ifstream og initierer med iname som argument
 - Merk at konstruktøren her også vil prøve å åpne filen. Ofte vil en se mer eksplisitt åpning og lukking av filer med funksjonene `open()` og `close()`, men læreboka anbefaler å benytte denne kompakte metoden, da den reduserer sjansen for feil (Forklart i PPP side 351-352)

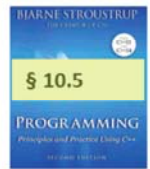
Opening a file for writing

An ofstream is an ostream for writing to a file

```
36 cout << "Please enter name of output file: ";
37 string oname;
38 cin >> oname;
39 ofstream ofs{ oname }; // ofstream is an "output stream to a file"
40                        // defining an ofstream with a name oname
41                        // opens the file with that name for writing
42 if (!ofs) error("can't open output file ", oname);
```

❑ Dette blir helt tilsvarende, bare for skrijving til fil

Example, reading a file



- File with a sequence of pairs representing hours and temperature readings

```
20 struct Reading { // a temperature reading
21     int hour;    // hour after midnight [0:23]
22     double temperature;
23 };
24
25 vector<Reading> temps; // create a vector to store the readings
26
27 int hour;
28 double temperature;
29 while (ist >> hour >> temperature) { // read
30     if (hour < 0 || 23 < hour)
31         error("hour out of range"); // check
32     temps.push_back(Reading{ hour, temperature }); // store
33 }
```

0	20.7
1	20.6
2	19.3
3	18.2
q	

16

Course TDT4102 – «C++ w/FLTK» - Lecture 7

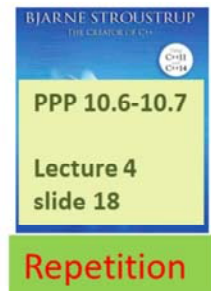
NTNU

Kodelinjer

- (20-23) Deklarerer en klasse (struct) for å lagre en temperatur-måling
- (25) Deklarer vector for å lagre disse
- (29) Så lenge while løkken klarer å lese time og temperatur fra fila (ist --- deklarasjon av den er ikke vist her) så vil betingelsen for løkka være sann (ist >> returnerer sann), straks vi møter f.eks. bokstaven 'q' så vil ist >> returnere usann fordi 'q' ikke er et heltall
- (31) kaller lærebokas (std_lib_facilities.h) sin funksjon error for å varsle at en ulovlig verdi er lest inn for time. Ingen kontroll for gyldig verdi av temperatur

I/O error handling

- Sources of errors
 - Human mistakes
 - Files that fail to meet specifications
 - Specifications that fail to match reality
 - Programmer errors
- `iostream` reduces all errors to one of four states
 - `good()` // the operation succeeded
 - `eof()` // we hit the end of input ("end of file")
 - `fail()` // something unexpected happened
 - `bad()` // something unexpected and serious happened



❑ Menneskelige feil kan f.eks. være tastefeil

Example: integer read “failure”

- Ended by “terminator character”
 - 1 2 3 4 5 *
 - State is **fail()**
- Ended by format error
 - 1 2 3 4 5.6
 - State is **fail()**
- Ended by “end of file”
 - 1 2 3 4 5 end of file
 - 1 2 3 4 5 Control-Z (Windows)
 - 1 2 3 4 5 Control-D (Unix)
 - State is **eof()**
- Something really bad
 - Disk format error
 - State is **bad()**

- ☐ Når vi sier «State is fail()» så mener vi at tilstanden er fail(), og det betyr stream-objektets medlemsfunksjon fail() vil returnere sann.
- ☐ Tilsvarende for de andre

User-defined input: `operator>>`



```
3 struct Date {
4     int day, month, year;
5 };
6 istream& operator>>(istream& is, Date& dd)
7     // Read date in format: ( year , month , day )
8     {
9         int y, d, m;
10        char ch1, ch2, ch3, ch4;
11        is >> ch1 >> y >> ch2 >> m >> ch3 >> d >> ch4;
12        if (!is) return is; // we didn't get our values, so just leave
13        if (ch1 != '(' || ch2 != ',' || ch3 != ',' || ch4 != ')') { // oops: format error
14            is.clear(ios_base::failbit); // something wrong: set state to fail()
15            return is; // and leave
16        }
17        dd = Date{ y,m,d }; // update dd
18        return is;          // and leave with is in the good() state
19    }
20
```

19

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

- ☐ Å definere input-operatoren `>>` er ofte mer krevende, det blir ofte en øvelse i feilhåndtering (Kalles også operator overloading)
- ☐ Dette eksemplet er fra PPP side 365, men med en litt enklere klasse `Date`
- ☐ Kodelinjer forklart
 - (6) Deklarasjon av `operator>>` for et `Date` objekt. Vi må ha referanseoverføring av `dd` siden hensikten er å lese fra inputstrømmen `is` og oppdatere `dd` objektet
 - (8-19) Kodeblokk som implementerer operatoren
 - (11) Her leser vi inn ett tegn som lagres i `ch1`, deretter ett heltall som lagres i `y`, ett tegn som lagres i `ch2` osv. Man antar formatet beskrevet i kommentaren i linje (7)
 - `is` vil ha verdien usann hvis lesingen i linje 11 feilet
 - (13) innlesingen gikk bra, men nå sjekker vi om de riktige skilletegnene er på plass, hvis de ikke er det så har vi format-feil (Noen ville ha foretrukket paranteser rundt hver av disse fire tegn-sammenlikningene)
 - (14) kaller medlemsfunksjonen `clear()` for å sette tilstanden til `is`. Her setter vi den til `fail()`-state ved å gi `ios_base::failbit` som argument
 - (17) det gikk bra og vi oppdaterer dato-objektet. Merk at vi her tilordnet et (navnløst) `Date` objekt til parameteren `dd`. **Tilordning (Eng. assignment) av objekter er vanlig** (Det er nærmere diskutert i 9.7.2 og 14.2.4)

Overview Lecture 7

- Repetition
 - Overloading operator<<
 - Larger example `cleanAir_V1.cpp` (later in the lecture)
- Associative arrays `<map>`
- static member variables
- `istream`, `ostream`, files
- User defined input operator
- Example `cleanAir_V1.cpp` (first version)
- More on output, and file modes



Example: `cleanAir_V1.cpp`

- Case: Air Pollution Sensors in a city
 - Example to demonstrate many **C++** elements lectured so far
 - Repetition
 - A small step towards realism (larger programs)



- Et bittelitt større eksempel der vi vil trekke inn ulike elementer dere har lært og demonstrere bruk

Example: `cleanAir_V1.cpp`

- Application
 - Air Pollution Sensor (APS) units located on a city map
 - Visualize air quality during 24 hours
 - Named locations
 - Many simplifying assumptions
- C++ demonstrated in version 1 (V1)
 - APSunit as class
 - enum class APSstate
 - `map<APSstate,string>` **and** `map<string,Color>`
 - get- and set-functions
 - Non-const static member variable (See yellow box in slide above)
 - Class Image from Graph_lib
 - Overloading output operator<<
 - (Reading sensor descriptions from file)

Air pollution

- The five most common components measured by APS units:
 - ozone, particulate matter, carbon monoxide, sulfur dioxide, nitrous oxide
- Assumption:
 - All can be represented by an `unsigned int`
- Source: https://en.wikipedia.org/wiki/Air_pollution_sensor
- An APS unit can be in one of the following states:

```
enum class APSstate {unknown = 0, planned, calibration,  
booting, ok, warning, bad, malfunc, flaky};
```

23

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

☐ På norsk

- Ozon
- Svevestøv (partikler)
- Karbonmonoksid (CO)
- Svoveldioksid (SO₂)
- Nitrogenoksid (NO_x)

☐ Dette er tenkte tilstander for en APS enhet

- Ukjent --- tilstand er ikke kjent
- Planlagt --- det er planlagt å sette opp en APS enhet på dette stedet
- Kalibrering --- enheten er under kalibrering/innkjøring
- Booting --- oppstart av programvare på enheten, f.eks. etter skifte av batteri el.l. kortvarig tilstand
- OK --- OK luftkvalitet
- Advarsel --- faretruende luftkvalitet, luften bør overvåkes
- Dårlig/ille (Eng. bad) --- meget dårlig luftkvalitet
- Malfunc = feilfunksjon --- noe feil med enheten
- Merkelig/upålitelig (Eng. flaky) --- enheten oppfører seg rart, bør undersøkes

☐ Merk at detaljer om luftforurensing i dette eksemplet rimeligvis IKKE er en del av pensum.

```

23 class APSunit {
24     // These are private by default (in class), but you can also use key
25     APSstate state{ APSstate::unknown };
26     unsigned int ozone = 0; // the five most
27     unsigned int particulateMatter = 0; // we
28     unsigned int carbonMonoxide = 0;
29     unsigned int sulfurDioxide = 0;
30     unsigned int nitrousOxide = 0;
31     Point location{ 0,0 }; // The location as coordinates on the city map
32     const string name; // Name of APS unit display on map cannot be changed
33     string description; // Description of
34     const string nameTag; // Short three
35     const int unitSerialNo = 0; // serial number
36     static int sensorId; // must be initialized outside the class
37     const int myId = 0; // An unique identity number (id) assigned to this unit
38     Vector_ref<Shape> display;
39 public:
40     APSunit(int sno, string name, string tag, Point loc, string desc):

```

Class APSunit (part 1)

Member variables initialized to default values. Strings will be initialized to empty string ""

A non-const static member variable must be initialized outside the class

declaration

24

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

❑ Dette er et realistisk objekt med en rekke medlemsvariable. Klassebeskrivelsen er ikke komplett mhp. medlemsfunksjoner enda

- Bl.a. har vi enda ikke funksjoner for å kunne sette eller lese de 5 heltallsverdiene som er det vi (later som vi) måler

❑ Vi viser og forklarer her utdrag av koden

- De komplette kommentarer finner du i filen APSunit.h
- Som det fremgår av bruken av const er det en del av medlemsvariablene som ikke kan endres, og noen som kan endres. F.eks. skal vi kunne endre location, description og display
- Merk at nullstilling (initialisering) av en rekke av disse variablene (f.eks. state, location og name m.fl.) strengt tatt er unødvendige fordi vi har laget en konstruktør med disse parametrene (neste slide). Derfor er eneste måte å opprette et objekt av denne typen å benytte denne konstruktøren. Å likevel ha med disse initialiseringene er **defensiv programmering**, som er en god ting. De vil få effekt om vi skulle finne på å fjerne den konstruktøren, eller noen av de aktuelle parameterene i den.
- Medlemsvariabel display er forklart på neste lysark

○ Kodelinjer

- (36) **sensorid** er en ikke-const static medlemsvariabel, denne må initialiseres en gang i programmet som en global variabel, da static betyr en kopi delt mellom alle objekter av den klassen. **Hensikten med denne variabelen** er en slags tellevariabel som forteller oss hvor mange sensorer vi har og som dermed kan brukes til å avgjøre hvilket nummer neste sensor skal få

Class APSunit (part 2)

A vector of references to all graphical elements (Shape-objects) that visualize the state of an APSunit

```
38     Vector_ref<Shape> display;
39 public:
40     APSunit(int sno, string name, string tag, Point loc, string descr);
41     APSstate get_state() const { return state; };
42     string get_name() const { return name; };
43     string get_nameTag() const { return nameTag; };
44     int get_myId() const { return myId; };
45     bool set_description(const string s); // updates the description, return
46     void set_state(const APSstate s);
47     void attach(Graph_lib::Window & win); // to make the sensor visible
48 };
```

APSunit constructor declaration

25

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

□ Del 2 av klasse-deklarasjonen til APSstate

- get og set-funksjoner
- Attach brukes for å kople de grafiske elementer lagret i medlemsvariabelen display til vinduet for visning der
- (38) en Vector-ref av referanser til de grafiske elementer som er knyttet til hver måleenhet, for visualisering
- (40) deklarasjon av konstruktør
- (41-44) Typiske get-funksjoner, med inline definisjon (altså inne i klasse-deklarasjonen)
- (45-46) to set-funksjoner, bare deklarasjonen, så de er definert utenfor klasse-deklarasjonen
- (47) Medlemsfunksjon for kople grafikk-delen av vinduet til objektet

Class APSunit constructor implementation

Start of initializer list

Assigns unique values to
myId using static member
variable sensorId

```
28 APSunit::APSunit(int sno, string name, string tag, Point loc, string descr)
29 : location{ loc }, name{ name }, nameTag{ tag }, description{ descr },
30 unitSerialNo{ sno },
31 myId{ ++sensorId } // Note that the value of myId is not initialized
32 | // by an argument but by the static class member sensorId
33 {
34     display.push_back(new Rectangle{ loc, APSwidth, APSheight });
35     display[display.size() - 1].set_fill_color(
36         textToColorMap.at(stateColorTextMap.at(state)));
37     display.push_back(new Text{ loc, name });
38     display[display.size() - 1].set_color(Color::blue);
39     static_cast<Text&>(display[display.size() - 1]).set_font_size(20); // d
40     // display stores Shape& entries, Text is derived from Shape (child
41     // the function set_font_size we must force the Shape& to act as a
```



26

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

❑ Konstruktøren til APSunit er relativt komplisert

❑ Kodelinjer forklart

- (28-32) er parameterlisten og initialiseringsliste, fordelt over fire linjer for å få dette passelig inn på et lysark. Mange ville nok ha programmert det mye mer luftig
- Linje 31 er litt spesiell, her initierer vi myId med verdien til en annen medlemsvariabel sensorID etter at den er inkrementert. Det vi gjør her er å tilordne hvert eneste objekt av klassen APSunit et fortløpende nr som lagres i myId. En kunne f.eks. ha brukt akkurat samme teknikk for å telle opp hvor mange objekter av en gitt type som skapes i programmet.
- (34) navnløst rektangel som legges i display, størrelsen er gitt av to konstanter APSwidth og APSheight. Vil vises på kartet
- (35-36) brukket over to linjer for å få plass på slide, vi setter fargen til rektanglet med en verdi basert på tilstanden. I linje 36 bruker vi to ulike map for å finne fargen
 - De er forklart på følgende to lysark
 - Husk at display.size() - 1 alltid vil være indeks til siste element lagt til med push_back
- (37-38) legger et Text objekt til display, for å vise name i blå skrift
- (39) vi setter font-størrelsen til Text-objektet
 - Her må vi bruke noe som heter downcasting og det vil bli forklart straks vi har forelest begrepet arv mellom klasser

map<APSstate, string>

```
enum class APSstate {unknown = 0, planned, calibration,  
booting, ok, warning, bad, malfunc, flaky};
```

```
2  const map<APSstate, string> stateColortextMap{ ,  
3      {APSstate::unknown, "white"},  
4      {APSstate::planned, "white"},  
5      {APSstate::calibration, "light_gray"},  
6      {APSstate::booting, "mid_gray"},  
7      {APSstate::ok, "green"},  
8      {APSstate::warning, "yellow"},  
9      {APSstate::bad, "red"},  
10     {APSstate::malfunc, "black"},  
11     {APSstate::flaky, "cyan"},  
12     };
```

*For use when logging color
in text-window or log-file*

27

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

❑ APSstate er en enum class som gir tilstanden til en målenhet

- Det valgte verdiene her er «etter fantasi»
- Avbildningen («mappet») beskriver den farge vi ønsker å vise frem for hver tilstand, f.eks. OK luft som grønn, warning (advarsel om høy luftforurensing som gul, og bad (farlig luftforurensing) som rød

How to map state to FLTK color?

```
14  const map<string, Color>textToColorMap{
15      // for setting FLTK- color from text
16      {"white", Color::white},
17      {"light_gray", Color::light_gray},
18      {"mid_gray", Color::mid_gray},
19      {"green", Color::green},
20      {"yellow", Color::yellow},
21      {"red", Color::red},
22      {"black", Color::black},
23      {"cyan", Color::cyan},
24  };

display[display.size() - 1].set_fill_color(
    textToColorMap.at(stateColortextMap.at(state)));
```

28

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

- ❑ Som parameter i `set_fill_color()` må vi bruke riktig datatype som er `Color` definert i `graph_lib`
 - Ved å definere avbildningen (map) som øverst på lysarket kan vi slå opp fra state til riktig `Color` ved å gi via den tekstlige beskrivelsen som `stateColortextMap` gir oss med oppslag

Parts of the main program

```
11 int main() try {
12     #ifdef _WIN32
13         SetConsoleOutputCP(1252); // Needed for Norwegian letters (Windows)
14         SetConsoleCP(1252); // Makes printing æ, ø, å, Æ, Ø and Å possible.
15         // Remember to set file encoding to Nordic (ISO 8859-10)
16         // note modified line 30 and new line 31 in Makefile to avoid encoding-warnings
17     #endif
18     cout << "cleanAir bruker C++ for et bedre miljø!\n\n";
19     Point topLeft{ 200, 300 };
20     Simple_window win{ topLeft, winWidth, winHeight, cityWinTitle };
21     cout << "... laster bykart\n";
22
23     ifstream testFileExists{ cityFileName }; // opening file to check that it exists
24     if (!testFileExists) error("can't open input file ", cityFileName); // Remember that error (from
25     // we use error here since it allows us to report also the filename for the file we tried to
26     Image cityMap{ Point{0,0}, cityFileName }; // The program hangs if file is not found, therefore
27     win.attach(cityMap);
28
29     Vector_ref<APSunit> allSensors;
30     cout << "... leser inn sensorer\n";
31     initSensors(allSensors, sensorsFileName);
32     for (int i = 0; i < allSensors.size(); i++) {
33         allSensors[i].attach(win);
34     }
```

29

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

□ Noen kodelinjer forklart eller begrunnet

- (12-17) For norsk tegnsett i konsollet. Dette er forbløffende problematisk, og koden gjelder bare Windows. Man må også inkludere «Windows.h». Windows-spesifikk kode er lagt innenfor betinget kompilering. #ifdef håndteres av preprosessoren, og på Windows er _WIN32 definert og linje 13-16 blir kompilert. Det er mulig man på mac og linux også må lagre koden i UTF-8-format, og ikke ISO 8859-10 som vi trenger under Windows. Se for øvrig kodeeksempel nordicLetters.cpp på github
- (23-25) Sjekker at filen lagret i konstanten cityFileName kan åpnes.
- (26-27) Vet da at vi kan åpne filen for å opprette et Image objekt, og kopler det til vinduet
- (29) Deklarerer en Vector_ref allSensors som lagrer alle sensorene
- (31) initSensors er en funksjon som leser dem inn fra filnavnet oppgitt ved parameteren sensorsFileName
- (32-34) går igjennom alle sensorer og kopler hver enkelt til vindu med attach()

Apparatur for måling av luftkvalitet i Trondheim, en gave fra TDT4102 til Bartebyen, våren...

Time: 5:00h

Example: **cleanAir_V1.cpp**

```

C:\cppc2020\deanAir_V1\debu...
1 Samfundet SMF mid_gray
2 Pirbadet PIR white
3 Marinen MRN black
... Luftforurensingsstatus klokken 2 er vist
  (IKKE implementert enda)
1 Samfundet SMF light_gray
2 Pirbadet PIR cyan
3 Marinen MRN light_gray
... Luftforurensingsstatus klokken 3 er vist
  (IKKE implementert enda)
1 Samfundet SMF black
2 Pirbadet PIR red
3 Marinen MRN black
... Luftforurensingsstatus klokken 4 er vist
  (IKKE implementert enda)
1 Samfundet SMF yellow
2 Pirbadet PIR black
3 Marinen MRN cyan
  
```

Trondheim

Code demo

Questions?

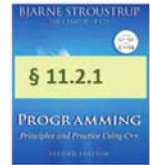
30 Course TDT4102 – «C++ w/FLTK» - Lecture 7 NTNU

Overview Lecture 7

- Repetition
 - Overloading operator<<
 - Larger example cleanAir_V1.cpp (later in the lecture)
- Associative arrays <map>
- static member variables
- istream, ostream, files
- User defined input operator
- Example cleanAir_V1.cpp (first version)
- More on output, and file modes



Integer output, decimal, hex or octal



- You can change “base”
 - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
 - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
 - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f
- They are «sticky» manipulators

```
cout << dec << 1234 << "\t(decimal)\n"
      << hex << 1234 << "\t(hexadecimal)\n"
      << oct << 1234 << "\t(octal)\n";
// The '\t' character is "tabulator"
```

Microsoft Visual Studio Debug Console

1234	(decimal)
4d2	(hexadecimal)
2322	(octal)

32

Course TDT4102 – «C++ w/FLTK» - Lecture 7

NTNU

- ❑ Dec, hex og oct er såkalte manipulatorer
- ❑ Sticky = klebrig
 - Når en av dem er brukt vil den være «klistret til» outputstrømmen inntil en av de andre brukes (de gjelder altså inntil videre)
- ❑ Dec, oct og hex kan brukes på samme måte for input (Se PPP 11.2.2)

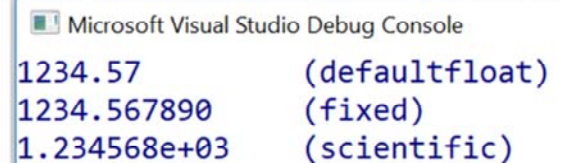
Floating-point output format

Manipulators

- **defaultfloat**: iostream chooses best format using n digits (default)
- **fixed**: no exponent; n digits after the decimal point
- **scientific**: one digit before decimal point plus exponent; n digits after

```
cout << 1234.56789 << "\t\t(defaultfloat)\n"
    << fixed << 1234.56789 << "\t\t(fixed)\n"
    << scientific << 1234.56789 << "\t\t(scientific)\n";
```

- Default precision is 6 digits
- **setprecision(n)** sets precision to n digits
- **setw(n)** sets width of field
 - also for integers



```
Microsoft Visual Studio Debug Console
1234.57          (defaultfloat)
1234.567890     (fixed)
1.234568e+03    (scientific)
```

File open modes

- By default, an **ifstream** opens its file for reading
- By default, an **ofstream** opens its file for writing
- The main alternatives:
 - **ios_base::app** *// append (i.e., output adds to the end of the file)*
 - **ios_base::binary** *// for binary files*
 - **ios_base::in** *// for reading*
 - **ios_base::out** *// for writing*
- A file mode is optionally specified after the name of the file:
 - **ofstream of1 {name1};** *// defaults to ios_base::out*
 - **ifstream if1 {name2};** *// defaults to ios_base::in*
 - **ofstream ofs {name, ios_base::app};** *// append*
 - **fstream fs {"myfile", ios_base::in|ios_base::out};** *// both in and out*

An fstream is an iostream for both reading and writing

- ❑ Forskjellen på append og vanlig skrivemodus er at sistnevnte skriver over innholdet i fila, det gamle innholdet slettes.
- ❑ (Læreboka side 389 beskriver noen fler detaljer som vi ikke vektlegger i dette faget)
- ❑ i sistel linje, `fstream fs ...` benyttes | som operatoren logisk eller for å kombinere de to flaggene `ios_base::in` og `ios_base::out`

Text vs. binary files

123 as
characters:



12345 as
characters:



123 as binary:



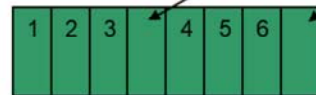
12345 as binary:



123456 as
characters:



123 456 as
characters:



In binary files, we use
sizes to delimit values

In text files, we use
separation/termination
characters



❑ Det er en rekke fordeler med tekstfiler framfor binærfiler

- du kan lese dem (uten noe fancy program)
- blir lettere å debugge programmet ditt
- portabelt
- det meste av informasjon kan representeres som tekst

❑ Vi vil komme tilbake til et eksempel på bruk av binærfil senere i kurset, dersom det blir tid og plass

stringstream

- A stringstream reads/writes from/to a string rather than a file or a keyboard/screen

```
double str_to_double(string s) {  
    // if possible, convert characters in s to floating-point value  
    stringstream is{ s }; // make a stream so that we can read from s  
    double d;  
    is >> d;  
    if (!is) error("double format error: ", s);  
    return d;  
}  
  
int main() {  
    double d1 = str_to_double("12.4"); // testing  
    double d2 = str_to_double("1.34e-3");  
    double d3 = str_to_double("twelve point three"); // will call error()  
}
```



stringstream, stringstream.cpp

- stringstream, istringstream, ostreamstream
- Assume ss is a stringstream
 - ss.str() returns a copy of ss's string
 - ss.str(s) sets ss's string to a copy of s

```
16      int i = 123;
17      ostreamstream oss;
18      oss << "Test " << i << " times";
19      cout << oss.str() << endl;
20      oss.str("New text");
21      cout << oss.str() << endl;
```

 Microsoft Visual Studio

```
Test 123 times
New text
```

37

Course TDT4102 – «C++ w/FLTK» - Lecture 7

 NTNU

- ☐ istringstream kan bare brukes for input
- ☐ ostreamstream bare output
- ☐ stringstream begge deler, se eksemplet