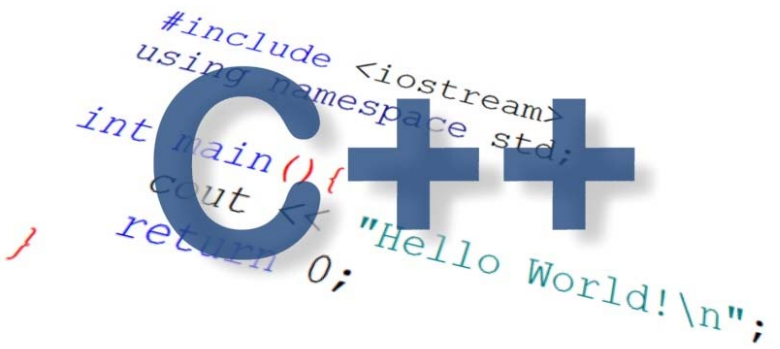
TDT4102 –
Prosedyre- og objektorientert programmering

Vi er mange: pakk godt sammen på benkeradene ! ☺

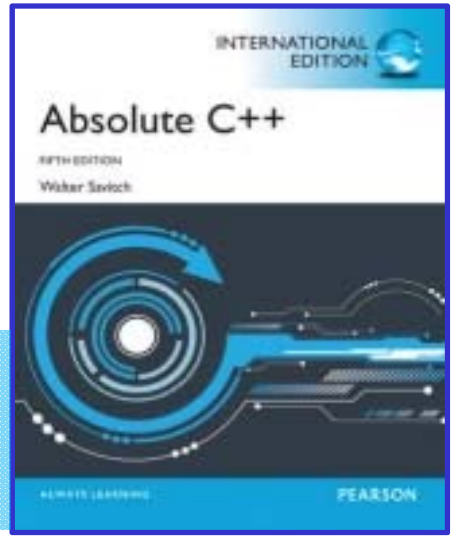


Grunnleggende C++ syntaks,
løkker, funksjoner

Innhold

- Grunnleggende C++
  - Litt intro i går: variabler, datatyper, …
  - I dag:
    - utrykk og tilordning, m.m.
    - inndata og utdata (lese inn og skrive ut verdier)
    - litt om løkker og forgreininger
    - Funksjoner, introduksjon

Kap. 1 – 4, 10.1



Uttrykk og operatorer

- Programmeringsspråket lar deg skrive uttrykk (Eng. expression) basert på operatorer og operander

```
3 + 4
2 * 4
```

- De vanlige aritmetiske operatorene +, -, *, /, %
- I programmeringsspråk brukes termen operator om mer enn de aritmetiske og boolske
  - Eks: tilordningsoperatoren =
- Ett uttrykk gir eksakt en verdi
- Alle uttrykk i C++ har returverdi

Sammensatte uttrykk

- Vi kan skrive mer komplekse uttrykk i en og samme setning:

```
2 + 4 * 20 / 2
```

- Del-beregninger foretas en for en
  - etter definerte regler (prioritet)
- Uttrykk i stedet for verdi
  - Et uttrykk kan benyttes der det er lovlig å ha verdien som uttrykket produserer som operand

Operatorer har innbyrdes prioritet

- Rekkefølgen på operatorene er ingen garanti for i hvilken rekkefølge deluttrykkene evalueres!
- Kompilatoren bestemmer en rekkefølge basert på regler for:
  - *Presendens: bestemmer hvilke operasjoner som skal gjøres først*
- **Bruk parenteser!**
  - *Uttrykk i parenteser evalueres først*
  - *I tilfellet parenteser inne i andre, evalueres innerste først*

Slå opp i boka Tabell (Display) 2.3 på side 79-80 eller bruk annen kilde for å finne de detaljerte reglene for presendens

Hva er uttrykkene..
Hvor mange finner du?


```
#include <iostream>

int main() {

    int hoyde = 5;

    int bredde = 8;

    int areal = hoyde * bredde;

    std::cout << areal;

    return 0;
}
```

Tilordning er også
en operator og vi har
her et tilordningsuttrykk

```
//Hva skrives ut?
cout << 2 + 10 * 3 + 6 / 3;
```

+ har lavere prioritet enn * og /

Utføres til slutt uansett om det står først eller sist

* og / har lik prioritet, men evalueres fra venstre til høyre

Dvs at * utføres først

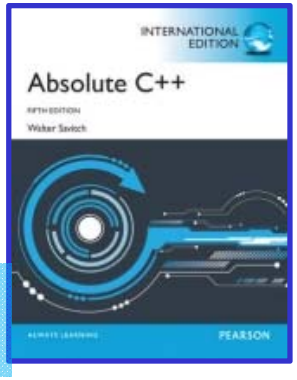
Vi kunne også skrevet: 2 + (10 * 3) + (6 / 3)

Da trenger vi ikke å huske reglene --- anbefales !

Regler for tilordning

- Må være kompatibilitet mellom høyre og venstresiden!
  - *Generell regel : du kan ikke tilordne en verdi av en datatype til en variabel som er av en annen datatype!*

- Vil i noen tilfeller gå bra på grunn av automatisk konvertering
  - Men slik automatisk konvertering kan gi problemer
  - Prøv å unngå det dersom du kan
  - (mer senere)



Side 52

To typer konvertering

- Implisitt (også kalt automatisk)
  - Divisjonsuttrykket resulterer i en skjult endring av heltall til flyttall

```
double y = 17 / 5.0;
```

- Eksplisitt type konvertering (casting)
  - Du spesifiserer hvilken konvertering som skal gjøres

```
double y = static_cast<double>(17) / 5;
```

- Husk at implisitt kasting også kan skje ved tilordning!

```
int x = 2.9;
```

Nyttige operatorer

- Inkrementering og dekrementering
  - Forenklet syntaks for ofte brukte operasjoner
- Inkrementering, ++

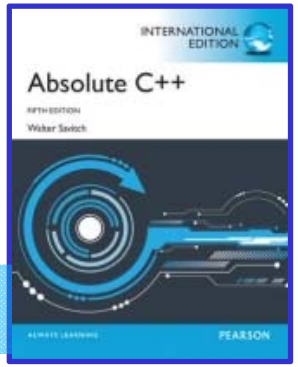
  a++; er ekvivalent med

  a = a + 1;

- Dekrementering, --

  a--; er ekvivalent med

  a = a – 1;



Side 79-80

Finnes også en rekke operatorer som er snarveier til sammensatte uttrykk +=, -=, /=

Kommentarer

- I programmering er det god praksis å skrive inn forklaringer
  - *For å gjøre koden mer forståelig og lesbar*
  - *Brukes med måtehold*

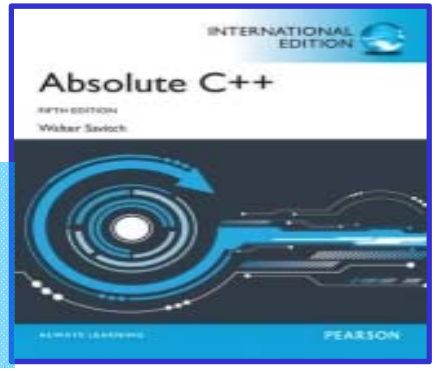
```
/* Dette er en kommentar som kan strekker seg over flere linjer
*/

// Dette er en kommentar på en linje

int x = y; //kommentarer kan også plasseres her
```

Inndata og utdata

- I dette kurset skal vi i første omgang kun skrive til konsollvinduet og lese fra tastaturet
- For å kunne benytte disse MÅ vi ha med denne linja i starten av kildekodefila

```
#include <iostream>
```

Side 56-62



std::cout og std::cin

- std::cout for output
  - *standard output (til konsollvinduet)*
- std::cin for input
  - *standard input (fra tastaturet)*

Hvorfor **std::**cout ?

- For å unngå navnekollisjoner i C++ programmering har **navnerom** (namespace)
- Funksjoner i C++ bibliotekene tilhører **std**
- Og må i utgangspunktet identifiseres med prefikset **std** kombinert med **::**
- For å slippe å bruke det fulle navnet til variabler og funksjoner i biblioteket kan vi ta med følgende i toppen av kildekodefila

```
using namespace std;
```

- Da kan vi droppe ”std::”

Operatorene vi bruker til input og output

- **<<** for å ”sende” en verdi til output
- **>>** for å «hente» en verdi fra input

```
int x = 0;
std::cout << "Skriv inn et heltall: ";
std::cin >> x;
std::cout << "Du skrev " << x << std::endl;
```

Boolske uttrykk i C++

- Egen datatype for boolske (logiske) verdier

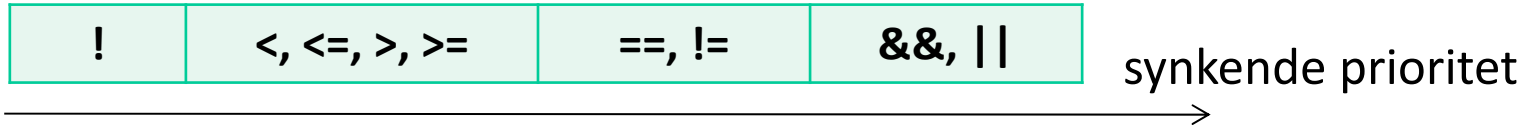
```
bool b = true;
```

- Uttrykk som returnerer boolske verdier
  - brukes for å kontrollere hvordan programmet skal oppføre seg
    - «Hvor det går», dvs. kontroll-flyt
- `true` eller `false` returneres når vi bruker
  - sammenligningsoperatorer (<, >, <=, >=, ==, !=)
  - logiske operatorer for and, or og not (&&, ||, !)

Operator prioritet m.m.

NB! Aritmetiske operatorer har prioritet over logiske.

- Operatorene har innbyrdes prioritet



- Short-circuit evaluering av boolske uttrykk
  - Evaluering avsluttes når utfallet av det sammensatte uttrykket er gitt:
    (x >= 0) && (y > 1)

hvis (x >= 0) gir false vil hele uttrykket returnere false uansett utfallet av (y > 1), programmet vil derfor unnlate å utføre siste ledd

- Boolske verdier og heltall
  - Boolske verdier er "kompatible" med heltallsvariabler
  - `false` = 0, `true` = 1 (eller i praksis tolkes tall != 0 som true)

NB! Noe du bør huske, men IKKE bruke!

if-else

- Parenteser rundt betingelsen
- else-delen kan utelates
- Hvis flere setninger skal utføres må du bruke blokker (krøllparenteser)

```
// finne største tall av x,y
if (x > y)
   largest = x;
else
   largest = y;
```

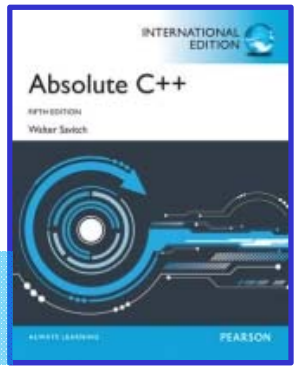
```
if (x > y) {
    largest = x;
    cout << largest;
}
```

Bruk tabulator og formatter med tanke på lesbarhet.
Fleksibelt hvor du plasserer krøllparentesene.
Godt råd: bruk krøllparenteser selv om det bare er en setning.

Mer om `if-else`

- `if-else` kan inneholde blokker av setninger
- Disse blokkene kan igjen inneholde andre `if-else` setninger
- Kan også ha flervalgs `if-elseif-else`

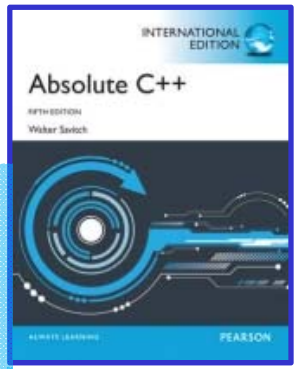
Side 90



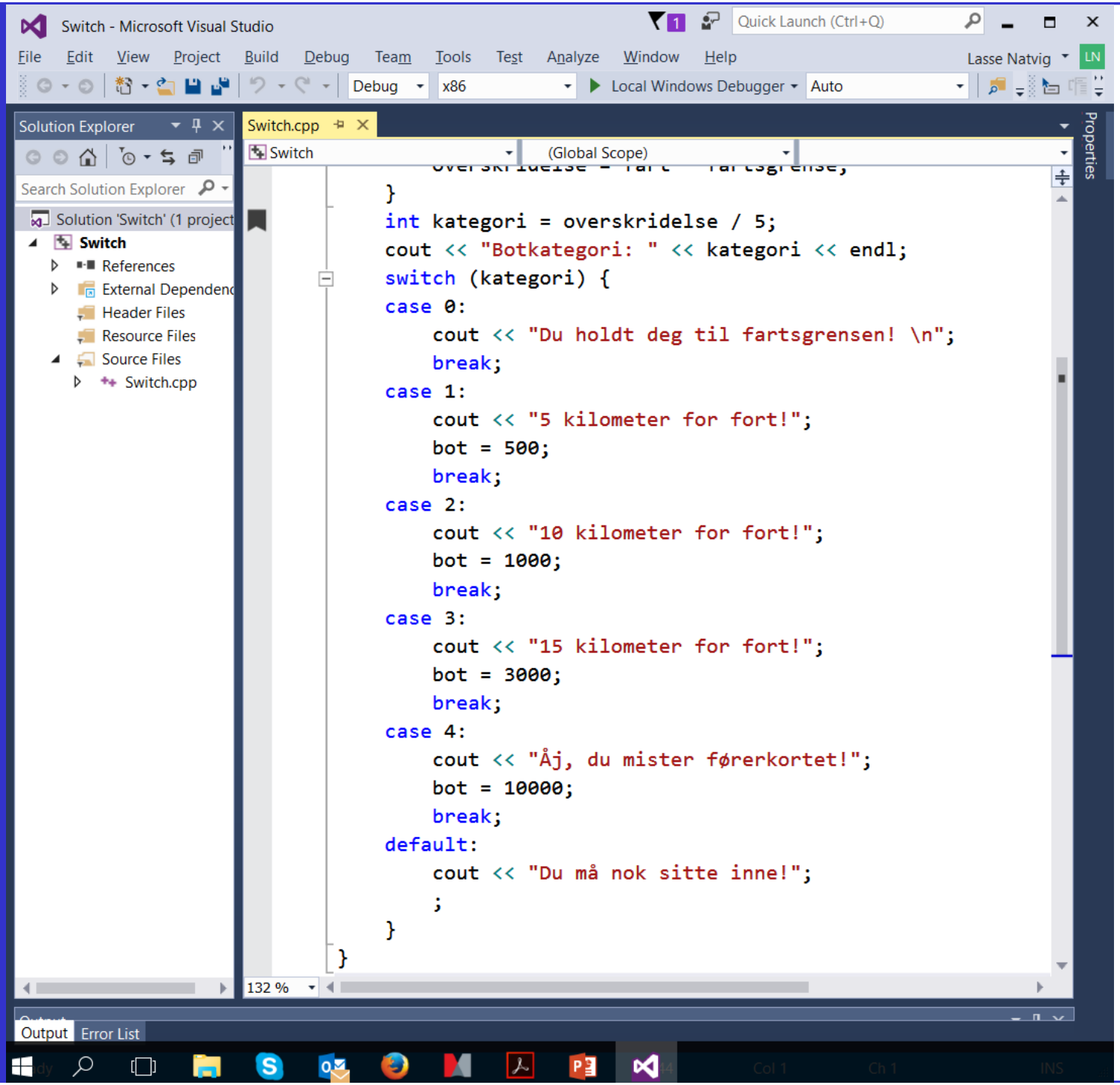
switch-setningen

- Hvis forgreiningen avhenger av en enkelt verdi kan du bruke switch-setning
- Basert på evaluering av en variabel eller uttrykk som gir en verdi av typen int
- Passer godt til “menyvalg”- lignende kode med enkle setninger for hvert case/tilfelle

Side 90-92



switch-setningen



switch og bruken av break

- Å utelate break er lov, men gir en spesiell oppførsel
- Utføringen starter på valgt verdi og alle setninger helt ned til første break utføres
  - uavhengig av om det kommer et nytt case før break!
- Hvis vi ønsker at flere case skal utføres
  - NB! break er ofte opphav til feil
  - Lovlig å ikke ha med break, men hvis vi glemmer å ta det med der vi trenger det gir det ofte helt feil "oppførsel"

Løkker

Dekkes av øving 1 ☺

- 3 typer løkker i C++
  - `while(){…}`
    - vanlig brukt løkke når betingelsen er boolsk
    - kan brukes til det meste
  - `do{…}while();`
    - mindre fleksibel (brukes mindre, men er nyttig av og til)
    - løkka utføres alltid minst en gang
  - `for(){…}`
    - naturlig for løkker hvor vi skal gjøre noe n antall ganger

while


```
int teller = 0;
while (teller < 3){
    std::cout << "Hei";
    teller++;
}
```

initialisering av teller utenfor løkka

betingelse for løkka

løkke-blokken: setningene som utføres om og om igjen til betingelsen er sann

Hvor mange ganger skriver vi ut ″Hei″?

Hei Hei Hei
0 1 2, og så avsluttes løkka

do-while


```
int teller = 0;
do{
    std::cout << "Hei";
    teller++;
} while (teller < 3);
```

initialisering av teller

løkke-blokken: setningene som utføres om og om igjen til betingelsen er sann

betingelse for løkka

Hvor mange ganger skriver vi ut ″Hei″?

Hei Hei Hei
1 2 3, og så avsluttes løkka

`while` vs. `do-while`

- Like men … en vesentlig forskjell
  - Forskjell på NÅR “when" delen evalueres
  - while: evaluerer før blokken
  - do-while: evaluerer etter blokken er utført
- **`while`**-varianten brukes i de fleste tilfeller

for-løkken

`for (initialisering; betingelse; oppdatering)`

```
for (int i = 0; i < 10; i++){
    std::cout << "Hei;
}
```

- Kompakt syntaks
  - Enkelt å deklarere og initialisere variabler, oppdatere variabler (telleren)
- Egner seg godt til løkker hvor vi skal
  - Iterere over et definert antall, inkrementell iterasjon etc.

break og continue

- Flytkontroll
  - Løkker bør være kontrollerte av betingelsesuttrykket !
  - Men vi har også en mekanisme for overstyre dette
- `break`;
  - tvinger løkka til å avslutte
- `continue`;
  - hopper over setningene som gjenstår
- Brukes kun hvis absolutt nødvendig!

Nøstede løkker

- Husk: ALLE gyldige C++ setninger kan være inne i løkkas kropp
- Inkluderer if-else setninger og andre løkker
- Dette gjør at vi kan lage nøstede løkker!
- Sterkt anbefalt å formattere med innrykk slik at det blir lesbar kode!

```
for (int i = 0; i < 10; i++){
    std::cout << i << ": ";
    for (int j = 1; j <= 10; j++){
            std::cout << (i *j) << "\t";
    }
    std::cout << std::endl;
}
```

Funksjoner

- Funksjoner er viktig i programmering
  - Bruke funksjoner, lage funksjoner, tenke funksjoner
- Funksjoner er ”byggeklosser”
  - en logisk enhet av funksjonalitet
  - forenkler problemløsing – løse mange små overkommelige problemer i stedet for et stort sammensatt problem
  - gjør koding lettere - skrive, lese, teste, vedlikeholde
  - gjenbruk er et viktig prinsipp i programmering

Hva er en funksjon?

- En enhet i programmet som:
  - tar imot data-input (argumenter)
  - prosesser dataene
  - produserer (returnerer) et resultat

- Men vi kan også ha funksjoner som
  - ikke tar input
  - ikke returnerer verdier
  - men en funksjon bør alltid gjøre noe
    - ellers er den overflødig...

Definisjonen av en funksjon

```
double sqrt(double);
```

- funksjonens identifikator som er navnet
  - (ofte et kompakt), men forståelige/forklarende navn
- parameterliste som definerer input
  - antallet argumenter og datatypen til hvert av argumentene
- returtype som definerer datatypen til returverdien

Egendefinerte funksjoner

- Å skrive egne funksjoner er en viktig del av å konstruere et program!
- Gjør koden ryddig, enklere å teste og finne feil, lettere å samarbeid om
- Dine funksjoner kan ligge i
  - samme fil som main()
  - eller i egne filer
  - (organisering av funksjoner i flere filer dekkes senere)

Deklarering, implementering og bruk

- Deklarering av funksjonen
  - Funksjonsprototypen (header)
  - Informasjon for kompilatoren slik at den kan sjekke at du bruker funksjonen riktig.
- Funksjonens implementasjon
  - Den faktiske koden som utføres
- Bruk av funksjonen
  - Aktivere funksjonen
  - Kalle funksjonen fra main() eller fra andre funksjoner

Vi skiller med andre ord mellom deklarasjon og implementasjon

Funksjonsprototypen

- SYNTAKS: <returtype> funksjonsNavn (<parameter-liste>);
- EKS:

```
int getGuess(int min, int max);
```

- Er en informativ deklarasjon for kompilatoren
- Forteller kompilatoren hvordan funksjonen skal brukes:
  - *hvilke typer som er lovlige som input og i hvilke uttrykk vi kan bruke funksjonen*
- Må stå før du bruker funksjonen i koden
  - *I første omgang plasserer vi denne over main()-linjen*
  - *Kalles også funksjonsprototyp*

Parameter-lista

- Deklarasjon av input til funksjonen
  - En funksjon kan ha tom parameterliste () eller ha en eller flere parametere (int x, int y)
  - Ved bruk må vi kalle funksjonen med riktig typer og antall argumenter

Retur-typen

- Alle funksjoner må deklareres med returtype
  - Enten en spesifikk datatype eller void hvis funksjonen ikke skal returne noe
- I funksjonsimplementasjonen må vi ha en return-setning hvor vi returnerer en datatype som stemmer med deklarert returtype
  - Funksjonen avsluttes når return kalles

```
return enVerdiEllerVariabel;
```

  - For "void funksjoner" trenger vi ikke return statement, men kan skrive return; hvis vi vil spesifisere at funksjonen skal avsluttes

C++ standard bibliotek

- C++ har bibliotek av funksjoner som vi ofte bruker
- Hvert bibliotek består av en samling funksjoner som "hører sammen"
- Disse er allerede kompilert og kan enten lenkes statisk eller dynamisk
  - *Statisk = programmet ditt inkluderer en kopi av funksjonene i bibl.*
  - *Dynamisk = funksjonene er ei egen fil som flere program kan bruke*
- Må bruke: #include <biblioteknavn>
  - *Gjør at kompilatoren inkluderer "spesifikasjonene" av funksjoner og typer i biblioteket (header-fil)*
  - *Mer om dette senere*

Finne ut hva bibliotekene inneholder?

- Oversiktlig liste i appendiks 4 i boka.
- Finnes egne bøker som gir detaljert beskrivelse av standardbiblioteket (med eksempler)
- Men du finner også gode oppslagsverk på nettet
  - http://www.cplusplus.com
  - http://www.cppreference.com
- Dokumentasjonen sier hva, men ikke hvordan
  - Forteller deg nok til at du kan bruke funksjonene
  - Men ikke hvordan funksjonen er implementert i detalj

Eksempel, enkelt testing

- Visual Studio
- Eksempel lagt ut under Its learning. Brukte ca. 25 minutter på det

Oppsummering

- Mange detaljer denne uka – egentlig bare repetisjon med ny syntaks og litt ”strengere” regler
- Det blir lettere å skjønne etter hvert, spesielt når en får erfaring fra å gjøre øvingene;-)