1. Grunnleggende C++

Leksjonen introduserer studentene til kurset og til C++. Etter å ha gått gjennom leksjonen skal de være i stand til å skrive enkle C++-programmer. De skal kunne bruke kontrollstrukturer, endimensjonale tabeller, nullterminerte tekststrenger og datafiler.

Skrevet av Else Lervik og Ole Christian Eidheim.

Eksemplene for denne leksjonen finner du her: https://gitlab.com/ntnu-iini4003/examples1

Støttelitteratur:

- Bjarne Stroustrup: A Tour of C++, Second Edition
  - Grunnleggende: kap. 1
  - Tabeller: kap. 1.7
  - Filstrømmer: kap. 10.7

Introduksjon til kurset og til C++

Velkommen til C++ for programmerere!

I løpet av semesteret skal vi arbeide oss gjennom størstedelen av den standardiserte delen av C++.

Fra før regner vi med at du har god kunnskap om objektorientering generelt og også erfaring fra et objektorientert programmeringsspråk, for eksempel C# eller Java. Ettersom de fleste utdanninger nå baserer seg på Java som programmeringsspråk, vil referansene i dette kurset gå til Java.

Du har ikke bare erfaring med programmering, du har også erfaring med å lære å programmere. Og du vet at det eneste som nytter er å kode, lage kode, endre kode og prøve ut kode. Men – i tillegg er det viktig å forstå hva som skjer. Å lære C++ handler ikke bare om å lære ny syntaks, det er også viktig å skjønne på hvilke områder C++ er fundamentalt forskjellig fra det språket du er vant med. På hvilke måter C++ oppfører seg forskjellig fra det du er vant med.

Kurset vektlegger det som skiller C++ fra Java. C++ har for eksempel en klasse `string` som ligner noe på Java-klassen med samme navn, og den vil vi selvfølgelig bruke. Men vi begynner med såkalte nullterminerte strenger som er en arv fra C, og som du heller ikke kan klare deg helt uten i C++. Det er mye vanskeligere enn å bruke klassen `string`, men så må du huske at dette er et kurs som skal lære deg det i C++ som ikke er som i Java … Og det er også slik at sannsynligheten for at ditt møte med C++ i arbeidslivet like gjerne blir å oppdatere eksisterende programvare som å lage ny. Ferdige klassebiblioteker (inkl. `string`) kom med standarden av 1998, og det gikk enda noen år før de første implementasjonene kom.

Alle først skal vi imidlertid se litt på tabeller i C++.

Du vil oppdage at C++ ikke passer på deg i like stor grad som Java. Ta en titt på følgende program:

```
// include refererer til biblioteker
#include <iostream> // innlesing/utskrift

using namespace std; // bruker standard navnerom, se cout nedenfor

int main() {
  int a;
  int b[3]; // en heltallstabell med 3 elementer
  double c;
  cout << "a = " << a << ", c = " << c << endl;  // kan skrive std::cout
  for (int i = 0; i < 5; i++) {
    cout << "i = " << i << " tabellelement: " << b[i] << endl;
  }
  return 0; // pga at main() er av typen int, kan sløyfes
}
```

Hvis du er Java-programmerer, legger du kanskje aller først merke til at `main()` ikke inngår i en klasse. Det betyr at du kan skrive C++-program uten å bruke klasser; det kan mest sammenlignes med å lage bare klassemetoder i Java. Koden refererer ikke til objekter.

Vi definerer de enkle variablene `a` og `c`, samt en tabell `b` med tre elementer. Uten å initiere variablene, skrives de ut. Utskriften kan bli som følger:

```
a = 256, c = 6.95322e-310
i = 0 tabellelement: 2304
i = 1 tabellelement: 3840
i = 2 tabellelement: 5376
i = 3 tabellelement: 6912
i = 4 tabellelement: 8448
```

Kompilatoren kan gi advarsler for linje 10, linjen der `a` og `c` skrives ut:

```
variable 'a' is uninitialized when used here
variable 'c' is uninitialized when used here
```

Hvilke konklusjoner kan vi trekke fra dette?

- Utskriften viser at variablene og tabell-elementene ikke er initiert til noen bestemt verdi.
- Vi får ingen advarsler eller feilmeldinger selv om vi skriver ut `b[3]` og `b[4]`, som ligger utenfor minneområdet til tabellen.

Dette lille eksemplet bør vise at vi må være påpasselige med alltid å initiere variabler, og vi må også selv passe på at vi ikke beveger oss utenfor det området i minnet som er satt av til en tabell.

C++-tabeller er ikke objekter, og de lages ikke med `new`, slik du er vant med. En tabell er rett og slett en samling variabler av samme type, som ligger ved siden av hverandre i minnet. Tabellen kjenner ikke sin egen lengde. Den enkelte variabelen når du ved å bruke indeksen, slik du er vant med fra Java. Navnet på tabellen gir deg adressen til begynnelsen av minneområdet.

I C++ 98 fikk vi klasse-templaten `vector`, og i C++ 11 fikk vi `array`. Førstnevnte ligner på `ArrayList` i Java, mens `array` i praksis fungerer som en Java-tabell, det vil si at tabellen er pakket inn i et objekt som kjenner sin egen lengde. Inntil videre skal vi imidlertid jobbe med "rå-tabellene", det vil si tabeller slik de alltid har vært, både i C og i C++.

Du vil nok oppdage at det kan være mye vanskeligere å finne feil i et C++-program enn i et Java-program. En av årsakene er selvfølgelig at det er nytt, men på grunn av forhold som påpekt foran, er det objektivt sett mye vanskeligere. Og da har vi ennå ikke begynt å se på pekere …

Du bør bruke litt tid på å lære deg å bruke debuggeren i det verktøyet du bruker. Det vil spare deg for masse tid og bekymringer senere.

Utviklingsmiljø, lærebok og C++ referanser

Det anbefales at du jobber i Linux og bruker utviklingsmiljøet juCi++ i dette faget. Dette utviklingsmiljøet er svært enkelt å bruke, og støtter samtidig de nyeste C++ standardene bedre enn andre utviklingsmiljøer. I tillegg er utviklingsmiljøet skrevet i nyere C++, og er av den grunn svært stabilt og lite ressurskrevende. Utviklingsmiljøet er åpen kildekode, og baserer seg på rundt 100 åpen kildekode biblioteker, og dermed har tusenvis av svært flinke utviklere bidratt til at dette utviklingsmiljøet har blitt en realitet. Du kan selv bidra ved å registrere eventuelle problemer som *issues* på nettsiden https://gitlab.com/cppit/jucipp.

Læreboken, Bjarne Stroustrup: A Tour of C++ Second Edition, er støttelitteratur til leksjonene dere finner på Blackboard. Det er ikke påkrevd at dere leser i læreboken, men det anbefales for å få mest ut av dette kurset. Hvis mulig, oppgir vi kapitler i læreboken som er aktuelle for det vi skriver om i leksjonene. Merk at boken er skrevet rettet mot utviklere som har erfaring fra andre programmeringsspråk, og i tillegg er rettet mot den nyeste C++ standarden c++17. Du kan selv aktivere den nyeste C++ standarden i juCi++ ved å endre `c++1y` til `c++1z` i `CMakeLists.txt` under en C++ prosjektmappe. Merk at da kreves det at du bruker en Linux distribusjon med nyere pakker som støtter den nyeste C++ standarden, som for eksempel Manjaro Linux. MacOS har også nyere pakker.

Det finnes også ressurser på nettet som kan være aktuelle å bruke når du programmerer C++. Det er hovedsaklig to nettsider som inneholder forklaringer med eksempler på innholdet i C++ standardbiblioteket: https://en.cppreference.com/w/ og http://www.cplusplus.com/. I juCi++, kan du gå til dokumentasjonen for funksjonen eller klassen du har markøren på (eller markert i *completions*) ved hjelp av Find Documentation i Source-menyen.

Nullterminerte tekststrenger

```
//
// eksempel.cpp
//
// include refererer til biblioteker
#include <cctype>   // char-behandling
#include <cstring>  // strengbehandling
#include <iostream> // innlesing/utskrift

using namespace std; // bruker standard navnerom

int main() {                // ikke void her!
  char text[5];             // en streng med maks lengde 5
  cout << "Skriv et ord: "; // utskrift, bruker <iostream>
  cin >> text;              // innlesing, bruker <iostream>

  for (int i = 0; i < strlen(text); i++) { // strlen() fins i <cstring>
    text[i] = toupper(text[i]);            // gjør om til store bokstaver, bruker <cctype>
  }
  cout << "Bare store bokstaver: " << text << endl;

  for (int j = 0; j < strlen(text); j++) {
    text[j] = tolower(text[j]); // gjør om til små bokstaver, bruker <cctype>
  }
  cout << "Bare små bokstaver: " << text << endl;
}

/* Kjøring av programmet:
Skriv et ord: test
Bare store bokstaver: TEST
Bare små bokstaver: test
*/
```

Sørg for at du får til å kompilere og kjøre dette programmet. Kildekoden finner du sammen med leksjonen. Du får et par advarsler ved kompilering, de skal vi komme tilbake til.

Studer koden sammen med kommentarene som kort forklarer hva som skjer på de enkelte linjene.

Programmet leser inn et ord og lagrer det i en såkalt nullterminert tekststreng (`char text[5]`). Prøv ut hva som skjer dersom du leser inn et ord med lengde større eller lik 5. Det går sikkert helt greit, kjøresystemet forsyner seg bare av den plassen som ligger utenfor tabellområdet. Men det går bare bra så lenge det ikke ligger andre data der, og det har du ingen kontroll med.

Strenger lagres i `char`-tabeller, og avslutningen av strengen markeres med tegnet '\0'. I tabellen `char text[5]` er det derfor plass til en tekst med maks lengde 4.

Programmet bruker tre biblioteksfunksjoner (en funksjon i C++ er det samme som en metode i Java), `strlen()`, `toupper()` og `tolower()`. Funksjonene anropes uten at de kvalifiseres med objekt- eller klassenavn. I praksis fungerer de som klassemetoder i Java.

Biblioteksfunksjonene er deklarert i filene `iostream`, `cstring` og `cctype`, og disse tas inn via preprosessordirektivet `include`. Symbolet `#` forteller at nå kommer det en preprosessordirektivet, og slike kommandoer utføres før kompilering ("pre" "processing"). I dette tilfellet settes innholdet i de nevnte filene inn i koden ("inkluderes") på den plassen der direktivet står, slik at funksjonsdeklarasjonene blir med i kompileringen.

Tilbake til advarselen som kommer ved kompilering:

```
comparison of integers of different signs: 'int' and 'size_t' (aka 'unsigned
long')
```

Dette gjelder linjene som ser slik ut:

```
for (int i = 0; i < strlen(text); i++) {
```

Returtypen fra `strlen()` er `size_t`. Og hva er så det? Det er et synonym for en ikke-negativ heltallstype, eksempelvis `unsigned int` eller `unsigned short`. Eksakt hvilken type som brukes er kompilatoravhengig og definert i en av include-filene.

Du kan unngå advarselen ved å skrive om setningen:

```
for (size_t i = 0; i < strlen(text); i++) {
```

Men det er nok slik at de aller fleste C++-programmere lever godt med denne advarselen og bruker int i eksempler som dette. Dog er læreboka ganske flink til å bruke `size_t`.

Lek deg gjerne litt med dette programmet før du går videre. Gjør endringer og utvidelser.

Enkel filbehandling

Følgende program leser tall fra en datafil. Programmet leser fram til filslutt. Studer kommentarene.

```
//---------------------------------------------------------------------
//
// tallfil.cpp
//
// Programmet leser tall fra fil og skriver summen av tallene til skjermen.
//
#include <cstdlib>
#include <fstream>
#include <iostream>

using namespace std;

int main() {
  const char filename[] = "tallfil.dat";
  ifstream file;       // definerer filvariabel
  file.open(filename); // åpner filen
  if (!file) {         // innfil kan brukes som et logisk uttrykk
    cout << "Feil ved åpning av innfil." << endl;
    exit(EXIT_FAILURE); // uthopp fra programmet
  }

  int number;
  int sum = 0;
  while (file >> number) { // leser fram til filslutt
    sum += number;
  }
  cout << "Summen er " << sum << endl;
  file.close();
}

/* Eksempel på kjøring:
Datafil:
3 4 5 5
17

Utskrift:
Summen er 34
*/
```

Hvor skal filen det leses fra plasseres? Du kan for eksempel sette opp absolutte sti i et filnavn ved å bruke /, for eksempel:

```
const char filename[] = "/home/ole/toerpot.dat";
```

Neste program viser utskrift til fil. Formateringen `setw(4)` betyr at tallet etterpå skrives ut høyrejustert over fire kolonner.

```
//---------------------------------------------------------------------
//
// toerpot.cpp
//
// Programmet skriver toerpotenser til fil.
// Alle toerpotenser mindre enn halvparten av
// maksimalverdien til "long int" skrives ut.
//
#include <climits>
#include <cstdlib>
#include <fstream>
#include <iomanip>
#include <iostream>

using namespace std;

int main() {
  const char filename[] = "toerpot.dat";
  ofstream file;
  file.open(filename);
  if (!file) {
    cout << "Feil ved åpning av utfil." << endl;
    exit(EXIT_FAILURE);
  }
  long int product = 1L;
  int exponent = 0;
  while (product <= LONG_MAX / 2L) {
    exponent++;
    product *= 2L;
    file << "2 opphøyd i " << setw(4) << exponent << " er "
         << product << endl;
  }
  file.close();
}

/* Filen ser slik ut:
2 opphøyd i    1 er 2
2 opphøyd i    2 er 4
2 opphøyd i    3 er 8
...
2 opphøyd i   28 er 268435456
2 opphøyd i   29 er 536870912
2 opphøyd i   30 er 1073741824
*/
```

Vi ser at vi leser og skriver mot fil akkurat på samme måte som vi leser og skriver mot tastatur og skjerm.

Å bruke `<<` og `getline()` om hverandre

Dette er ikke noe du trenger i øvingen i dag, men det tas med her da det ofte er en aktuell problemstilling som medfører mye frustrasjon.

Vi bruker `<<` til å lese inn ett ord (fra tastatur eller fra datafil). Ordet tolkes som tall eller tekst i henhold til datatypen til variablene vi leser inn i.

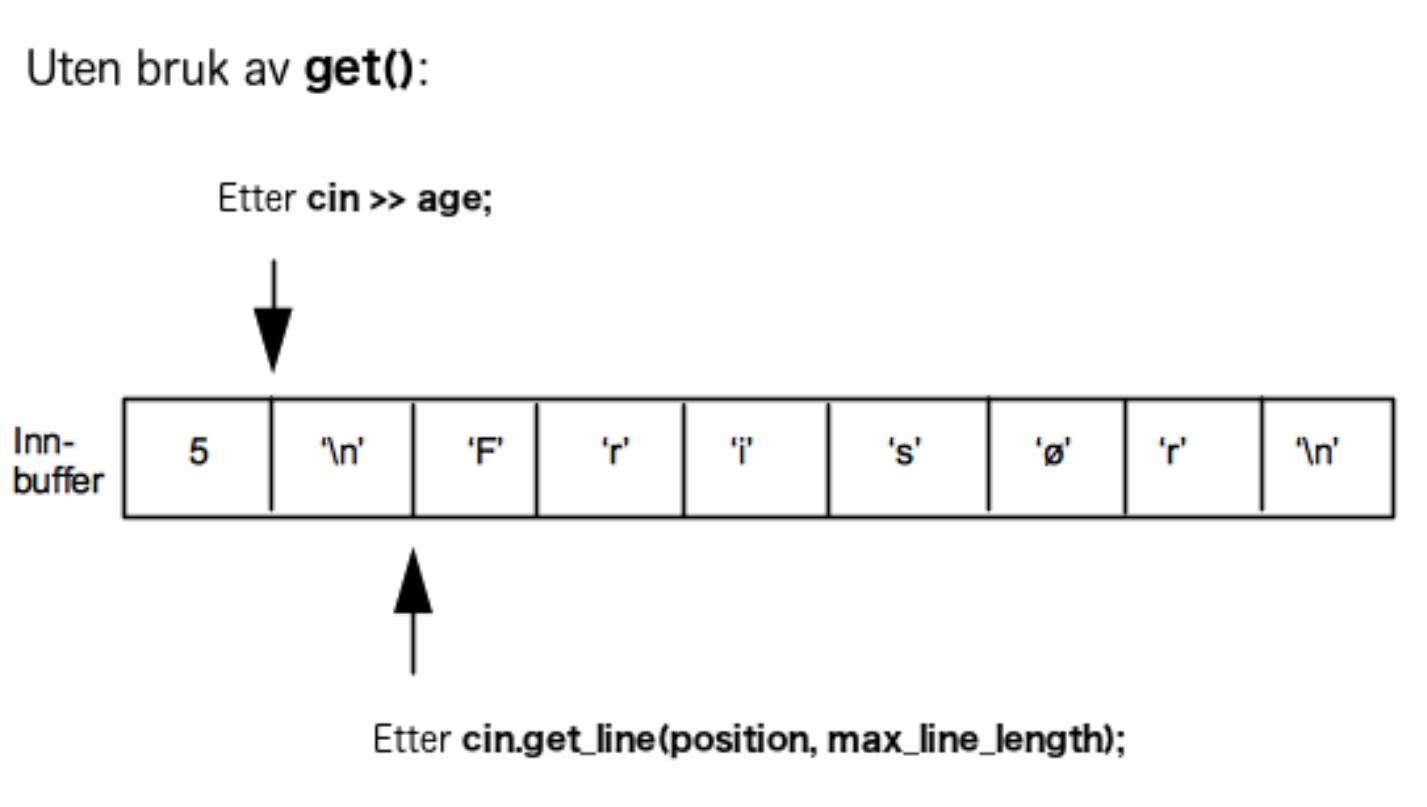
Vi bruker funksjonen `getline()` for å lese inn en linje. Eksempel:

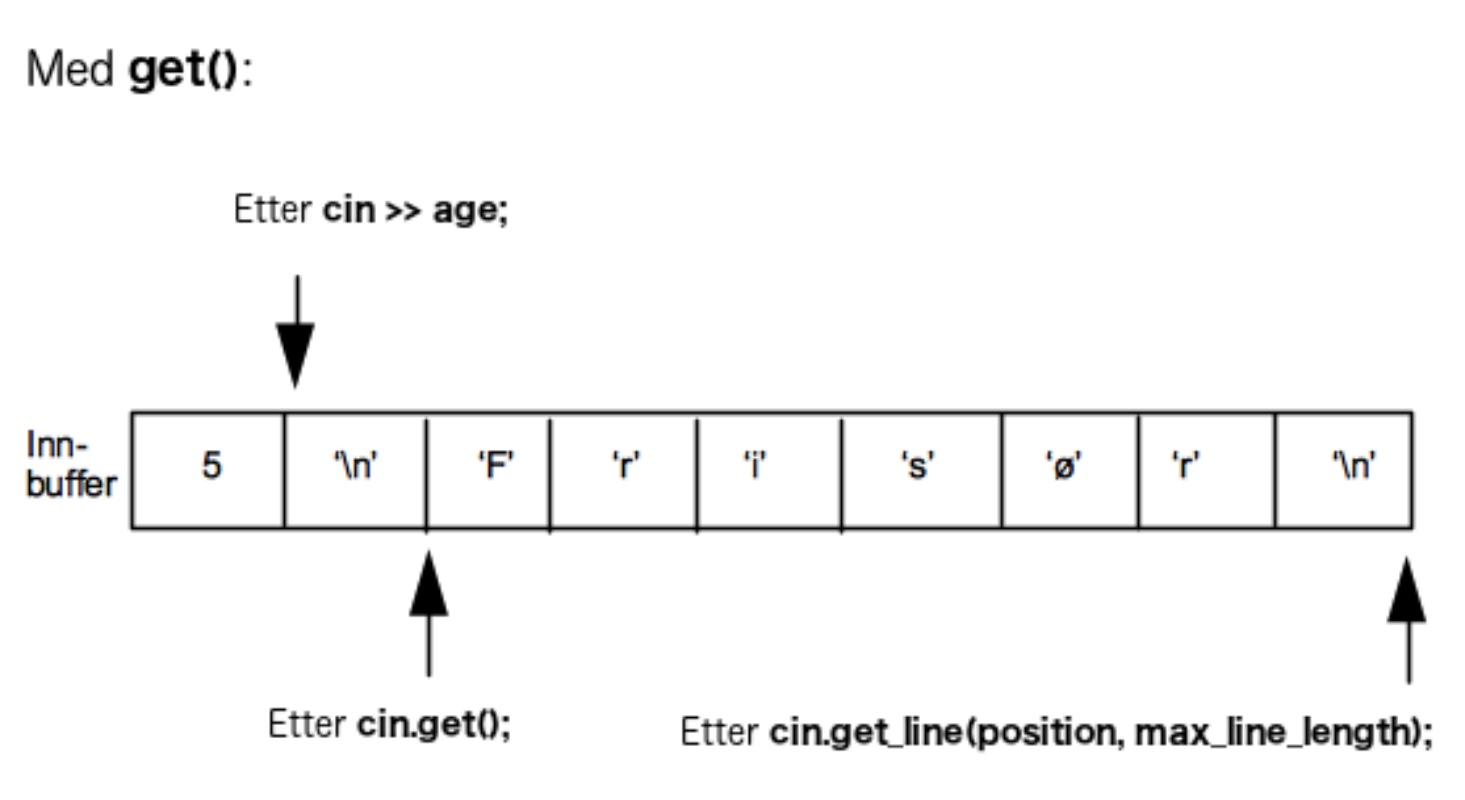
```
char text[100];
cout << ”Skriv en tekst: ”;
cin.getline(text, 100); // maks 100 tegn leses inn, inkl. ’\0’
```

Inndataene vil gjerne være en blanding av tall og tekst. Tallene, ofte flere pr. linje, ønsker vi å lese inn med `cin >>`, mens vi vil lese inn teksten linjevis med `cin.getline()`.

`>>` hopper over ordskiller, inkludert linjeskift. Funksjonen `getline()` søker etter linjeskift og avslutter innlesingen når et slikt påtreffes. Linjeskiftet leses altså ikke inn.

Øverst på figuren under er det vist hva som skjer hvis programmet etter å ha lest inn et tall (`age`) med `cin >>`, leser inn en tekst (`position`) med `cin.getline()`. Pilene viser hvor langt i inn-bufferet programkontrollen er kommet etter at de to setningene er utført. `position` blir tom.





Funksjonen `get()` henter et tegn i inn-bufferet, og gjør det dermed mulig å hoppe over tegn i inn-bufferet.

Programmet nedenfor viser eksemplet fra figuren satt inn i en sammenheng.

```
//---------------------------------------------------------------------
//
// person.cpp
//
// Prøver getline() og ignore()
//
#include <iostream>

using namespace std;

const int max_line_length = 81;

int main() {
  char name[max_line_length];
  char address[max_line_length];
  int age;
  char position[max_line_length];

  cout << "Navn: ";
  cin.getline(name, max_line_length);
  cout << "Adresse: ";
  cin.getline(address, max_line_length);
  cout << "Alder: ";
  cin >> age;
  cin.get(); // Tar bort newline ('\n')
  cout << "Stilling: ";
  cin.getline(position, max_line_length);

  cout << name << endl << address << endl << age << endl << position << endl;
}

/*
Kjøring av programmet:

Uten get():

Navn: Kari Ås
Adresse: Storgt 17, 7000 Trondheim
Alder: 26
Stilling: Kari Ås
Storgt 17, 7000 Trondheim
26

Med get():

Navn: Kari Ås
Adresse: Storgt 17, 7000 Trondheim
Alder: 25
Stilling: Frisør
Kari Ås
Storgt 17, 7000 Trondheim
25
Frisør
*/
```

Programmet er kjørt med og uten `get()`-setningen. Det første eksemplet viser kjøring uten `get()`. Etter ledeteksten Stilling: begynner utskriften. Setningen

```
cin.getline(position, max_line_length);
```

fører bare til at tegnet `\n` i inn-bufferet hoppes over. `position` blir en tom tekststreng, noe vi ser av at kjøreutskriften ikke viser noe etter at alderen (26) er skrevet ut.