Ting der skal huskes C++

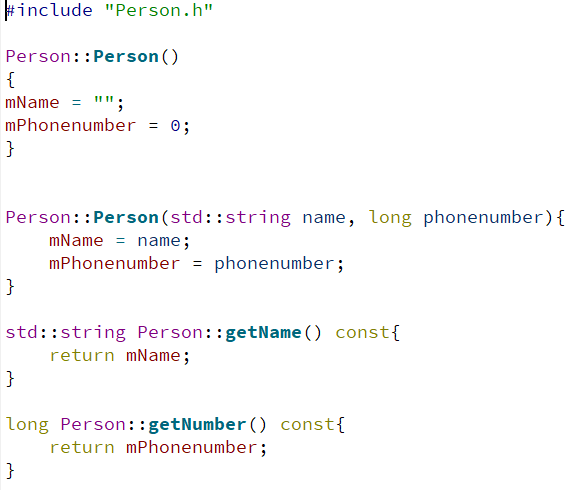
Opsætning af C++-projekt i QT-creator:

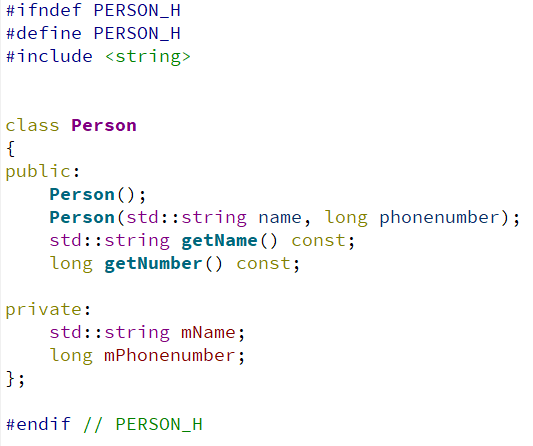
Start QT-creator 🡪 filer 🡪 ny fil eller projekt 🡪 ikke Qt projekt 🡪 C++-projekt 🡪 vælg filplacering 🡪 vælg navn 🡪 qmake byggesystem 🡪 mingW kit 🡪 slut

Lave en klasse i QT-creator:

Åbn et projekt 🡪 filer 🡪 ny fil eller projekt 🡪 C/C++ (filer og klasser) 🡪 C++ klasse.

Eks.

Header-fil: kildekode:



HUSK at **#include ”klassenavn.h”** (headerfilen) i kildekoden og skriv **klassenavnet::funktionsnavnet** når funktionerne skal implementeres i kildekoden.

Smarte biblioteker at include i programmer:

**<iostream>:**

Står for at tage input fra bruger f.eks. std::cin >> var1 >> var2; eller std::getline(std::cin, var);

Her tager getline alt det der skrives i terminalen og lægger det i variablen var, hvorimod std::cin tager til første mellemrum.

Kan desuden printe til terminalen f.eks. std::cout << ”Noget sejt tekst blabla, og en var = ” << var << std:endl;

Kan også bruges til fejlbeskeder f.eks. std::cerr << ”Error: \ ”error message\” ” << std::endl;

**<array>:**

Henter arrays ned så man kan arbejde med dem.

Initialisering: std::array<datatype, nrElements> name; ELLER -||- {} (alle initialiseret til 0) ELLER -||- {1, 2, 3, 4} (list initialization);

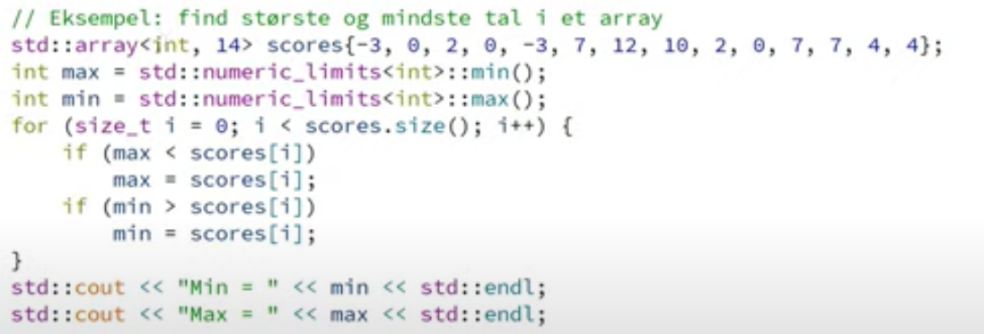
Flerdimensionelt array: std::array<datatype, antalSøjler>, antalRækker> name{{{1, 2, 3, 4, 5}, {1, 2, 3, 4, 5}, {1, 2, 3, 4, 5}}}; (matrix med 5 søjler og 3 rækker i dette eksempel hvor hver række har elementerne 1, 2, 3, 4, 5.)

Element for flerdimensionel array: name[i][j];

Første element name[0];

Man bruger ofte std::size\_t til at gennemløbe arrays i forløkker.

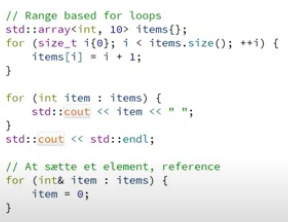
name.size() returnerer størrelsen på arrays.

Her findes min og max i et array.

Størrelsen af arrayet skal angives i compiletime og kan derfor ikke angives ”during the run” (her kan der bruges vektorer).

Angives arrayet med ”static” foran i en funktion gemmes det igennem hele programmets levetid.

Rangebased for-loop ved gennemløbning (her skal en reference bruges til at ændre værdierne):



Man kan sortere arrays vha. funktionen std::sort(name.begin(), name.end()); (sort tager 2 parametre: et start-indeks og et slut-indeks, som her er start og slut). (**HER INKLUDERES <algorithm>, for at funktionen virker**).

Man kan desuden lave standard arrays ved at sige hhv. int array[10 = {….}; og int array[10][20] = {…}

**<vector>:**

De er ligesom arrays, men dynamiske, altså at de kan ændre størrelse samtidig med at programmet kører.

En vektors elementer skal ligge lige efter hinanden i hukommelsen.

Initialisering: std::vector<datatype> name(størrelse) Eller man kan bruge list initialization ligesom ved arrays {1, 2, blabla}

Funktionen name.size() returnerer størrelsen.

Funktionen name.at(13) returnerer kun værdi hvis der er mindst 13 elementer. Her kan man i stedet bruge name[13] som ikke giver fejlbesked.

Man kan tilføje elementer med funktionen: name.push\_back(value);

Man kan slette det sidst tilføjede element med funktionen: name.pop\_back();

Kapaciteten (ikke størrelsen men pladsen som kan ændres dynamisk) kan findes ved name.capacity();

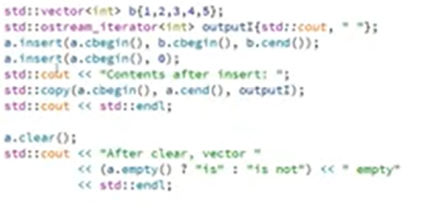
Name.reserve(antalElementer) kan reservere en kapacitet til en vektor (bruges ikke så meget, men er der).

Name.shrink\_to\_fit() bruges til at skærre kapaciteten ned til antallet af elementer i vektoren.

Man kan også slette elementer i en vector f.eks. ved at bruge nedenstående:

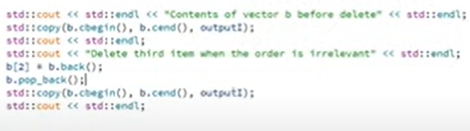


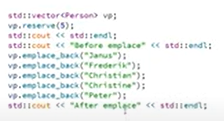
Man kan også indsætte en vektors elementer i en anden vektor ved at bruge nedenstående:



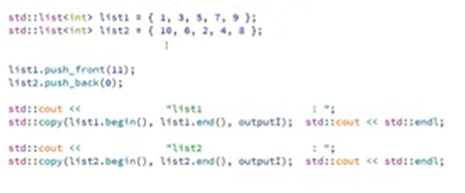
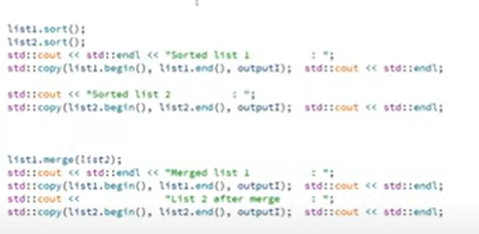
Clear() og en iterator med erase() er det samme.

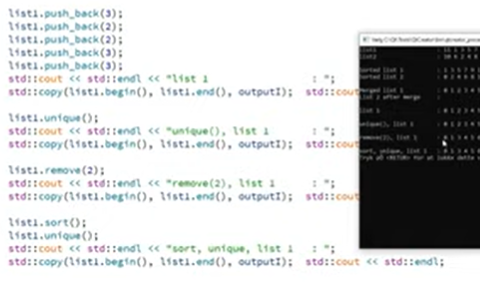
Man kan slette et element ved at gøre nedenstående:

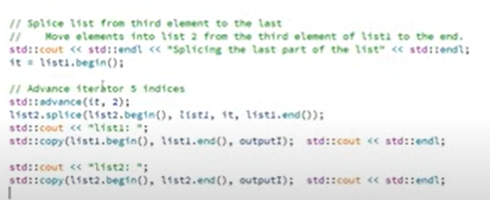


* Man kan på en og samme tid oprette objekter og tilføje dem til en vektor med funktionen emplace\_back(parametre til constructoren):
* 

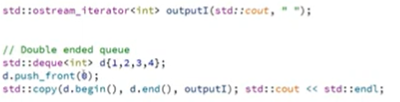
**<list>:**

* Giver adgang til lists som er gode til at indsætte elementer i starten af containeren.
* 
* Man kan sortere listerne ved at kalde name.sort();
* Man kan sammensætte to lister ved at kalde name1.merge(name2); Her lægges name2’s elementer ind i name1-listen i sorteret rækkefølge. Her fjernes alle elementerne fra name2.
* 
* Name1.unique(); fjerner flere på følgende ens elementer.
* Name1.remove(<datatype> tal); fjerne alle elementer som er = tal.



* ved at bruge splice kan man indsætte en liste i en andens liste på en iterators placering:
* Her bruges advance til at forøge iteratoren med 2.
* 
* Et andet eksempel på brugen af splice. Her har den lidt flere parametre.
  + Her vil der lægges elementer ind i liste 2’s start, de skal tages fra liste 1 fra iteratorens placering it og fortsætte til liste1 slutter.
* 

**<deque>: double ended queue.**

* Er lidt det samme som en vektor bortset fra at den er god til både at få indsat elementer i starten og slutningen.
* 

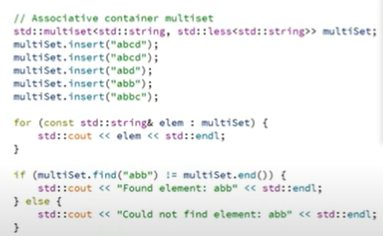
**<string>:**

Bruges til at inkludere strings.

Initialisering: std::string varnavn = ”noget”;

**<set> og multiset;**

* Et multiset er en container som sørger for at al data hele tiden er sorteret.
* Her er std::string datatypen og sorteringsmetoden er std::less<std::string>;
* .find(element) kan bruges til at finde et element.



**<map> : og multimaps:**

* Container med par:
* Nedenfor viser .first og .second hhv. første og anden key, altså stringen og integeren.
* .equal\_range(tal) viser elementet med tallet som key.
* 

**<limits>:**

Bruges til at finde minimum og maksimum størrelse for en datatype:

Eks. Std::numeric\_limits<datatype>::lowest();

Std::numeric\_limits<datatype>::min();

Std::numeric\_limits<datatype>::max();

**<iomanip>:**

Bruges til at sætte antal betydende cifre som skal vises i terminalen f.eks.

Std::cout << std::setprecision(5) << 0.11111111 << std::endl; (printer 0.11111 i terminalen)

Funktionerne std::left, std::right, std::internal kan også bruges her til at vælge hvor man placerer ting i terminalen.

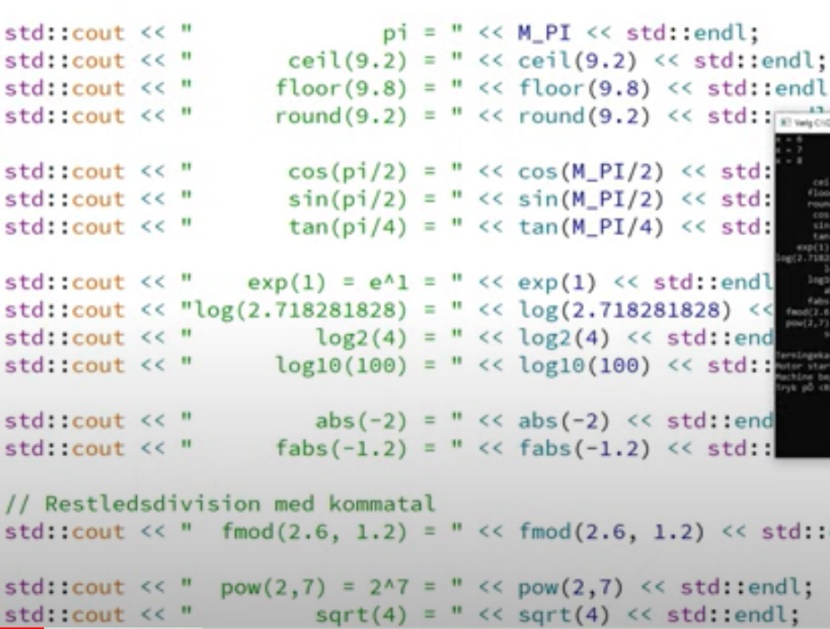
Desuden kan funktionen std::setw(n) til at fortælle at man vil bruge n karakterer til at printe sin string.

f.eks. std::cout << std::setw(20) << ”hi” << ”hi ” << ”hi” << std::endl; (printer : hi hi hi)

std::showpoint sørger for at setprecision printer det samme antal decimaltal efter kommaet.

**<cmath>:**

En masse matematiske funktioner bla:



**<random>** og **<ctime>** (tilfældighedsgenerator):

To klasser der tilsammen kan lave en tilfældighedsgenerator som ikke kan forudsiges.

Initialisering:

std::default\_random\_engine generatornavn{static\_cast<unsigned int>(time(0))};

fordeling af tal (her i integers):

std::uniform\_int\_distribution<unsigned int> random{minTal, maxTal}

output for én gangs brug af generatoren:

random(generatornavn);

**<error-messages og eceptions> : include <stdexcept>**

Man kan lave sin egen fejlmeddelelse ved først at lave en klasse der arver fra en exception:

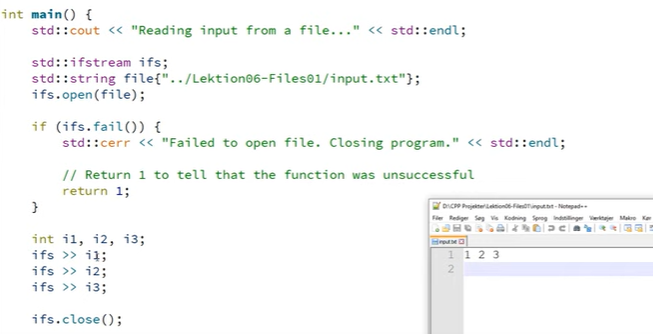


* For at kaste en exception skal der skrives: throw ExceptionName og den skal senere fanges med catch(std::exception& e);
* 
* Der kan desuden fanges flere forskellige fejl per enkelt try.
* Det smarte med try and catch er at programmet kører videre i stedet for at crashe hvis man prøver at gøre noget man ikke må og det fanges af en catch.
* 
* Man kan desuden også bruge std::cerr til at udskrive fejlbeskeder.
* De objekter der oprettes i en try-blok nedlægges.
* Fejlbeskederne kan genkastes ved dette.
* 
* Her bruges det nederste throw til at sende den fejlbesked den har fanget videre. (rethrow).
* 
* Her ses så at den kan fanges.
* Når man kaster en exception leder koden tilbage til den nærmeste catch.

**<fstream>(skriv & hent fra filer) og <sstream>:**

Arbejde med input og output til og fra filer.

Man skriver filveje relativt til den mappe/den fil hvor programmet køres fra. Eks. Nedenunder, gås en gang længere ud fra mappen (..) og dernæst åbnes /lektion…/input.txt. (det er altså filplaceringen.



Her oprettes et objekt kaldt ifs af typen std::ifstream. Funktionen . open() åbner filen og .close() lukker.

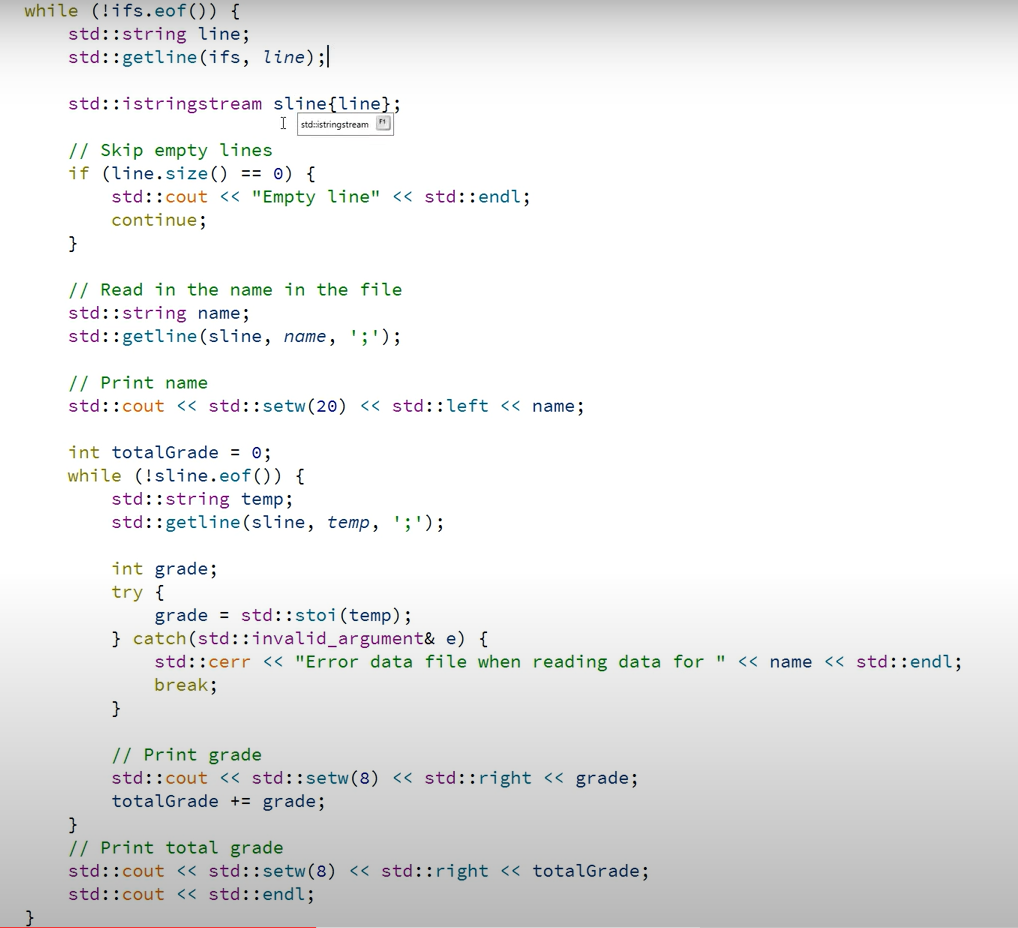


Her i eksemplet læses alle integers og lige så snart der ikke er tale om en integer så stopper programmet med at læse.

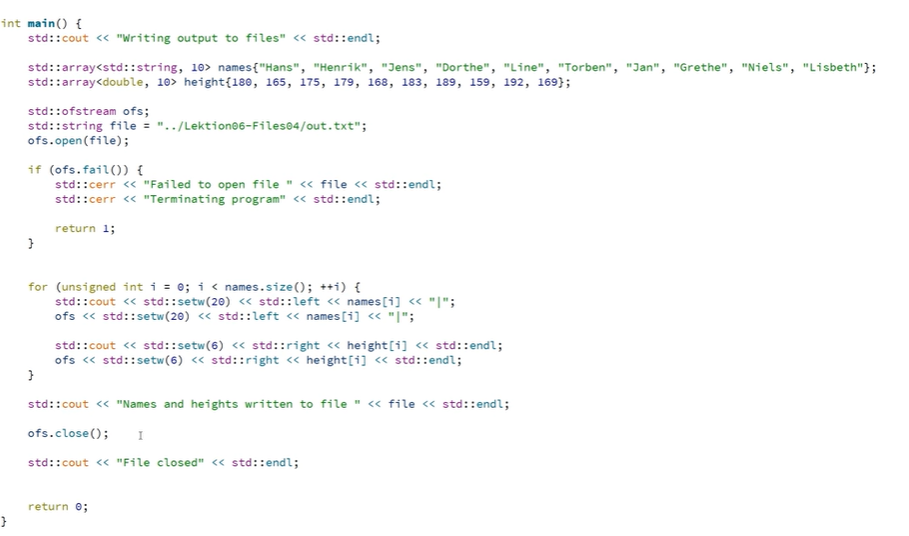
Læsning af filer med blandede datatyper:

<https://www.youtube.com/watch?v=ixENUoQ7h9c>

Hvis data er separeret med semikolon i filen brug da:



Hvis der skal skrives i en fil bruges fstream og eksempel herpå er:



Skrives til filen via objektet ofs af typen std::ofstream.

Operatorpræcedens:

1. ()
2. \* / %
3. + -

Boolske udtryk:

1. !
2. < > !=
3. &&
4. ||

Specialtegn:

\” (anførelsestegn)

\t (tabulator, stort mellemrum)

\n (linjeskift)

\r (går tilbage til starten af linjen og overskriver)

\b (går et tegn bagud i linjen)

Switch:

En konstruktion til at udføre kode alt efter hvilken værdi en variabel har.

Switch(varnavn) {

Case 1:

kode;

break;

case 2:

kode;

break;

default:

kode;

}

Casting (ændring af datatype)

Static\_cast<datatype>(denVarManVilÆndreDataTypePå);

Defaultargumenter:

Double funktion(double price = 0, double number = 0){

}

Referencer:

Ændringer på variablene påvirker også det stykke data som de refererer til.

Int a = 2.0;

Int& b = a (reference til a);

Eks. ( b++; Std::cout << a < <” ” << b << std::endl; (printer: 3 3)

Pointers:

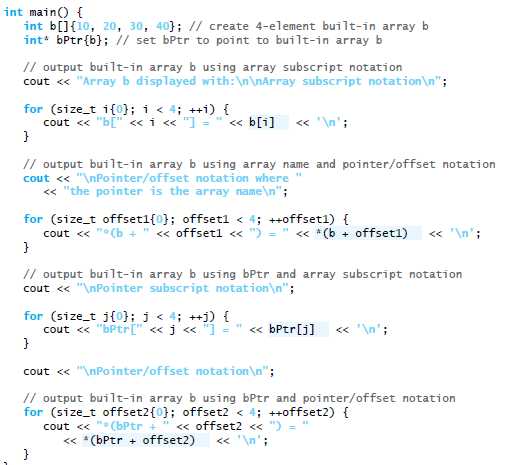
Variabel der peger ind på hukommelsens adresse på computeren.

Eks. dataType \*p1 = &var; (her peger pointeren p1 på adressen for variablen var.

\*p1 = værdien for det som pointeren peger på.

P1 = adressen for det som pointeren peger på.

Ændrer man \*p1 ændres også værdien for det som p1 peger på.

Ved arrays kan man gennemgå dem på forskellig vis vha. pointers.

Standard indbyggede arrays:

Datatype name[size] = {3, 4, 2, 1, 4, …};

Stream-manipulators:

Std::dec (decimaltal)

Std::hex (hexadecimal)

Std::oct ( octaltal)

Std::setbase(integer) (skriv et tal med base integer eks. 2 for binær osv.

Std::setw(integer) (antal pladser der skal bruges)

Std::fixed (hele tallet med 6 cifre efter kommaet)

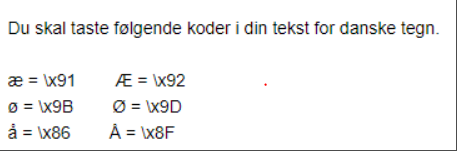
Std::scientific (et tal foran kommaet og E+blabla)

Std::defaultfloat (3 cifre efter kommaet)

Std::setprecision(integer) (integer antal cifre efter kommaet hvis det er med fixed, ellers er det totalen).

Std::showpoint (viser altid cifre efter kommaet selvom de ikke har nogen matematisk værdi for tallet eks. 1.00 uden: 1 med: 1.00)

Std::boolalpha og std::noboolalpha ( kan vise om boolske variable er true eller false)

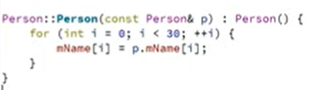


**CopyConstructor: (Objekt obj1 = obj2)**

Kopier alle member-variable til det andet objekt og sørger for at data ikke deler memoryplacering (smart ved brug af pointere som dataobjekter).

Man bruger den ved at sige Objekt obj1 = obj2; (Her er obj2 initialiseret først)

Eksempel på implementering:



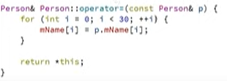
Vær her opmærksom på at mName-arrayet skal laves først (og den laves i constructoren som også her kaldes øverst ( : Person()). Hvis ikke det var der skulle man lige skrive mName = new char[30];

**Assignment-operator: (Obj1 = obj2)**

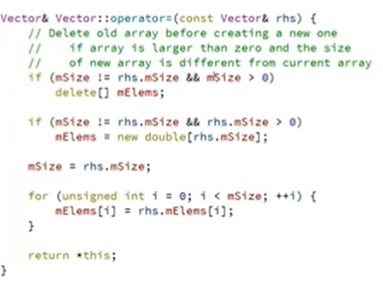
Næsten det samme som copyconstructor bortset fra at objektet er initialiseret i forvejen.

Man bruger den ved at sige obj1 = obj2;

Eksempel på implementering:



Endnu et eksempel på implementering hvor de to objekter ikke har samme størrelse på deres membervariabel som er et array af double-værdier:



Nedenfor ses et eksempel hvor der bruges dynamisk allokeret data:

Her slettes data nemlig og oprettes på ny. Det samme vil ske ved copy-constructoren.



Og nedenfor ses endnu et eksempel:

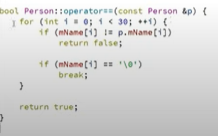


**Comparison-operator: (obj1 == obj2)**

Man kan finde ud af om to objekter er ens.

Man kan f.eks. bruge den ved at spørge if(obj1 == obj2) {std::cout << ”nice” << std::endl;}

Eksempel på implementering:

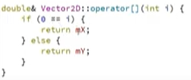


**Firkant-operator: obj[i]**

Gør så man kan bruge en firkant til at tilgå noget data. I dette eksempel bruges [] da der er tale om elementer i en vektor. Double& viser datatypen som skal returneres hvilket er mX og mY her.

Man kan bruge den ved f.eks. at sige obj[1] = 4.0;

Eksempel på implementering:



**-og + operator: (Object obj = obj1+obj2)**

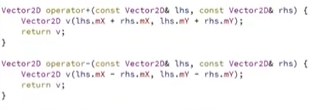
Her kan man lægge 2 vektorer sammen for at lave en ny vektor.

Her er funktionerne angivet som friend i headerfilen da både rvalue og lvalue er parametre.

I headerfilen:



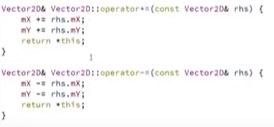
I cpp-filen:



**-= og += operator: (obj1 += obj2)**

Man ændrer et objekt ved at lægge et andet til.

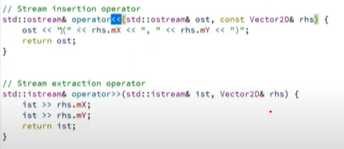
Eksempel på implementering:



**Stream-insertion-operator (print): << og Stream-extraction-operator(indsæt): >>**

Hhv. printer og indsætter information i objekterne.

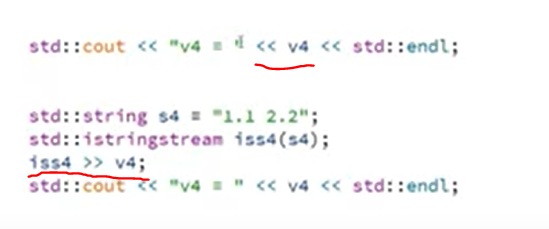
Eksempel på implementering.



Disse skal også implementeres som friend og skrives derfor 2 gange i headerfilen som vist nedenfor.



De kan kaldes som vist nedenfor:

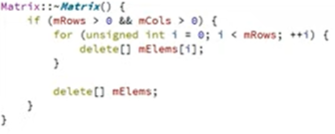


**Destructor:**

Sørger for at rydde op i hukommelse og kaldes automatisk af compileren. Her skal man slette pointere og andre membervariable, dog ikke hvis de er simple datatyper.



Eksempel ved destruction af multidimensionelt array:



**Multidimensionelle arrays med pointer:**

Eksempel:

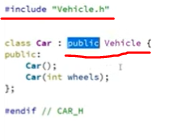
Double \*\*array = new double\*[10] ; (10 = Antallet af rækker).

For (int i = 0; i < 10 antal; i++){ array[i] = new double[20];} (20 = antallet af søjler).

**Nedarvning:**

Moderklasse( r) deler sine attributter og metoder med børneklasserne.

* Sæt attributter i moderklasse til protected.
* Skriv :public ClassName efter den klasse der skal arve, her ses at Car arver fra Vehicle.
  + Husk desuden #include på de klasser der indgår.



* Hvis man vil anvende en moderklasses constructor kan man skrive (: constructornavn(evt.parameter)): (Her er det angivet explicit)



* Hvis man vil sikre at der er dynamisk binding til objekttyper indsætter man virtual foran funktionsnavnet på moderklassen og override efter funktionsnavnet i datterklassen:
  + Eksempel: virtual void method(); og void method() override; (header)
* Hvis man gerne vil have en abstrakt moderklasse kan man lave en pure virtual metode (det er nok bare at lave 1, for at der er tale om en abstrakt klasse):
  + virtual void pvMethod() = 0; (header)
  + Denne skal ikke implementeres i moderklassen men derimod i alle børneklasserne:
  + Her skal der da stå: void pvMethod() override; (header).
  + Denne skal også implementeres i børneklassen og kan f.eks. skrive ”Objekt metode” eller noget andet.
* Dynamic\_cast<Object\*>(obj) tjekker om et objekt (obj) kan konverteres om til datatypen Object\*.
  + Dette er smart hvis man f.eks. har en vektor med objekt\* og man vil tjekke hvor mange objekter der er af hhv. f.eks. biler, fartøjer osv. Så kan man skrive if(DET DER STÅR I STARTEN) så….
* Husk også at lave destructor virtual hvis man vil sikre at man sletter det rigtige objekt.
* Multipel nedarvning : man skriver f.eks. class AutonomusCar : public DenEneKlasse, public den AndenKlasse.
  + PROBLEM hvis de klasser der arves fra har samme funktioner da compileren da ikke ved hvilken en der skal bruges.
  + Man kan der kun skrives obj1.CLASS::method();

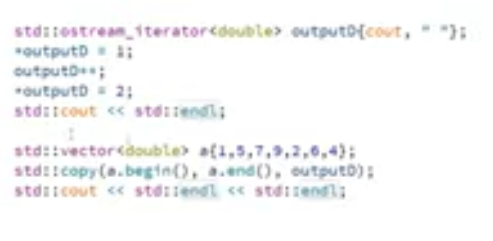
**Iterators: INCLUDE <iterator>:**

* Bruges til at pege ind på elementer i containers.



**Ostream\_iterator.**

* Printer de tal med datatypen her double som skrives til den med \*navn med her std::cout og indsætter et mellemrum imellem hvert tal.
* Husk at forøge iteratoren med 1 efter hver gang der printes.
* Nederst er en lille algoritme som printer en vector.



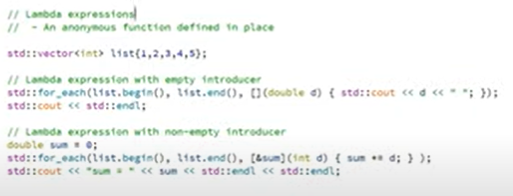
**Iterators til print af vektorer både forfra og bagfra:**

* De siger lidt sig selv.
* Const\_iterator bruges hvis elementerne ikke skal ændres.



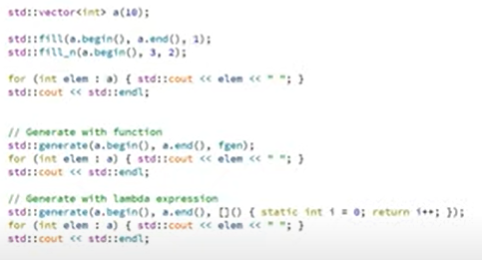
**Lambda-expression:**

* En kort funktion der ligger inde i koden.

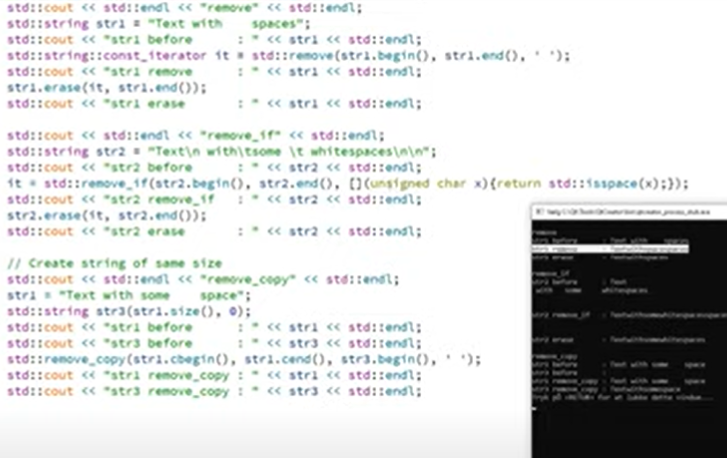


**<algorithm> til vektorer:**

* Algoritmer til at fylde en vektor.
* Fill(start\_position, slut\_position, hvad skal fyldes i alle elementerne.
* Fill\_n(start\_position, hvor mange elementer skal der fyldes i, hvilket tal skal der fyldes i).
* Geenrate(start, slut, funktion) (her kan der godt bruges en lambda-expression).



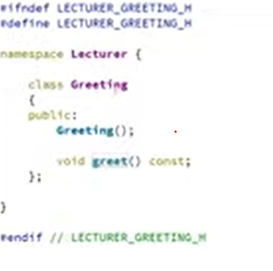
* Sammenlign elementer i to arrays:
* 
* .equal(start\_pos, slut\_pos, hvorfra skal der sammenlignes i et andet array).
* Mismatch kan fortælle hvilket element der ikke er ens som det andet.



* Std::remove(start, slut, hvilke tegn der skal fjernes)
* Std::remove\_if(start, slut, funktion)
* Std::remove\_copy = gør det samme som remove().

**Namespaces:**

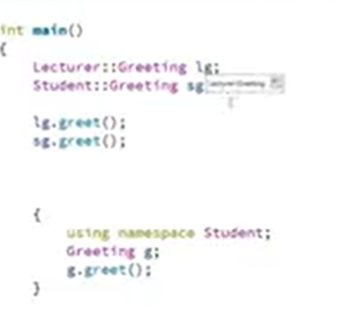
* Kan bruges hvis man f.eks. gerne vil kalde to eller flere klasser ved samme navn, men lade namespace være forskellig. Std:: er også et namespace.
* Nedenfor ses header-filen (til venstre) og cpp-filen (til højre).

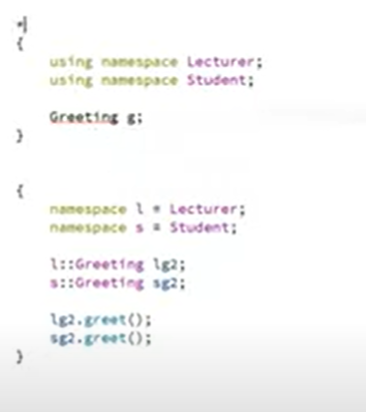


* **Vigtigt: ændr i include-guard så det følger ovenstående eksempel.**

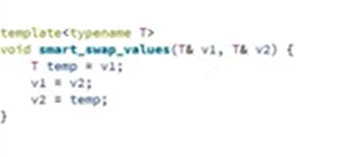


Nedenfor ses hvordan man bruger namespace.

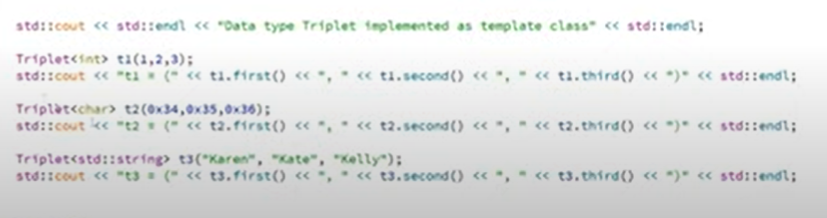


* **Bemærk. Aldrig brug namespaces i headerfil, eller to namespaces på samme tid.**
* Nedenfor ses at man også kan lave en shortcut til namespaces.
* 

**Template-funktioner:**

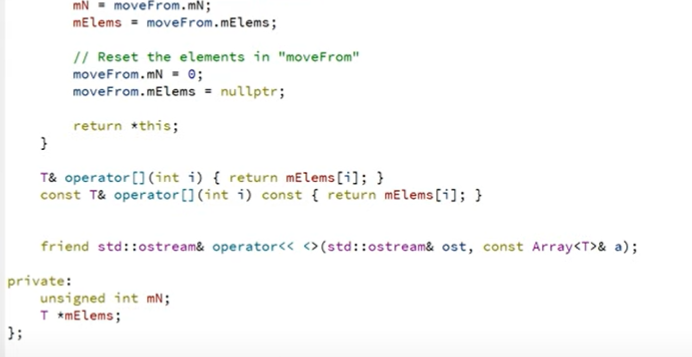
* En smart og hurtig klasse man kan lave i sit main-program, som har en udefineret datatype.
* 
* Her ses funktionen smart\_swap\_values.
* Her bruges T som symbol for datatypen.
* Nedenfor kaldes funktionen på hhv. to integers og to chars:
* 

**Template-klasse:**

* En klasse hvor man selv vælger datatypen:
* Nedenfor gives et eksempel med en klasse kaldt Triplet<datatype>:
* Nedenfor ses main-programmet.
* 
* Nedenfor ses headerfilen hvori funktionerne også er implementeret nederst.
* Der skal altså ikke være en cpp-fil.
* 

Hvis man gerne vil lave en friend-funktion i en template-klasse skal man forward deklarere klassen samt funktionen og deklarere den med et <> nederst i klassedeklarationen. Derudover skal man blot implementere den nederst i headerfilen som altid.



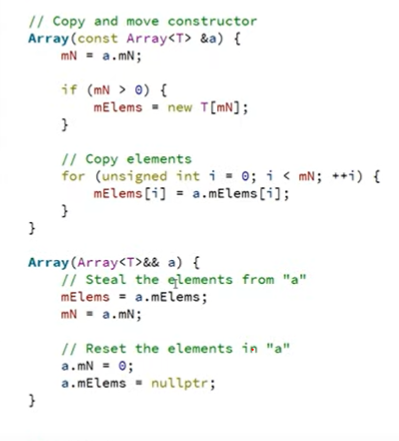


**move-constructor og move-assignment:**

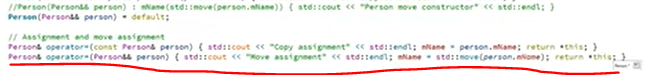
**Move-constructor:**

* Man flytter data fra et objekt til et andet
* Hvis man bruger både copy-constructor og move-constructor vil copy-constructoren benyttes. Derfor kan man implementere sin egen move-constructor. Se nedenfor.
* Benyt kun std::move i implementeringne hvis der er tale om ikke-simple datatyper såsom string.
* 
* Ovenfor ses move-constructoren. Bemærk dobbelt && og ingen const.
* Hvis man gerne vil bruge default-move-constructoren kan man skrive det der står nedenfor den røde linje.
* 
* Ovenfor ses brugen af move-constructoren.
* Hvis man gerne vil bruge default move-constructoren kan man skrive således:



* Ved lidt mere komplekse klasser, hvor der bruges pointere kan man gøre således med hvhv. Copy-og move -constructor:
* 

**Move-assignment:**

* Her kan man flytte data fra et objekt til et andet:
* 
* Nedenfor ses både copy-assignment og move-assignment.

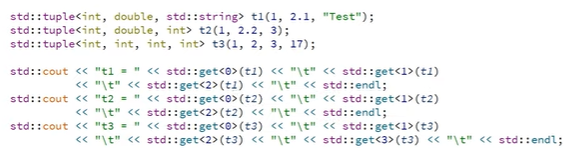


Nedenfor ses endnu et eksempel hvor der bruges dynamisk allokeret data.



**<tuple> :**

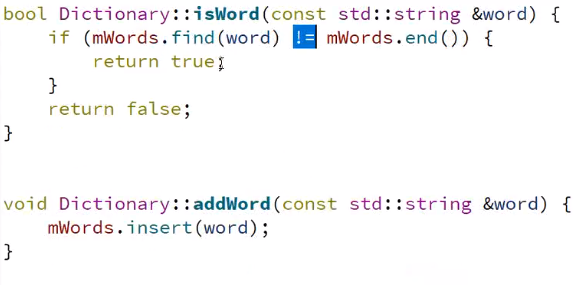
* Tupler er containers der kan indeholde 3 datatyper.



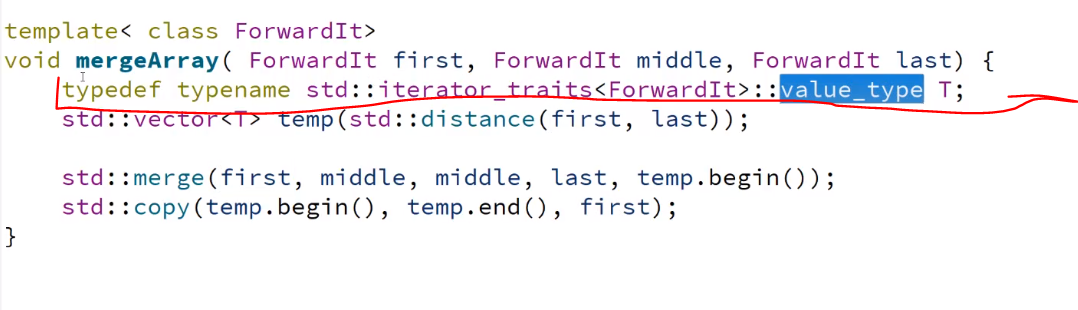
**Std::set<datatype> :**

Er en anden slags container.

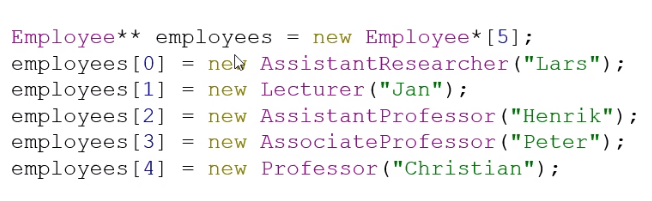
Er altid sorteret.



**Find ud af hvilken datatype en variabel er:**



**Array med pointere:**

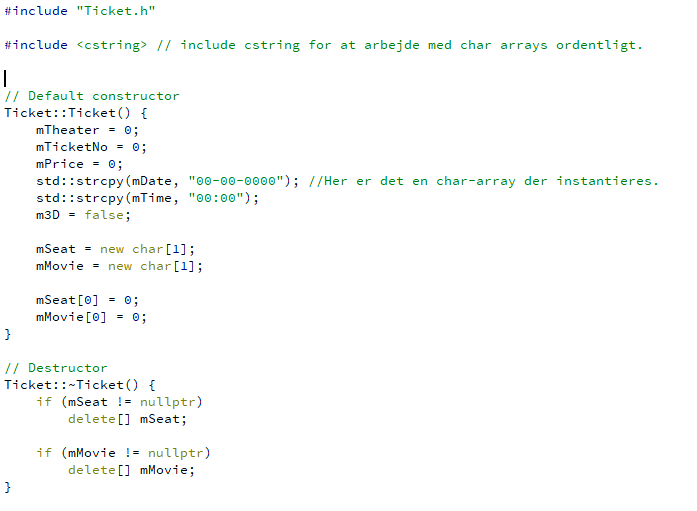


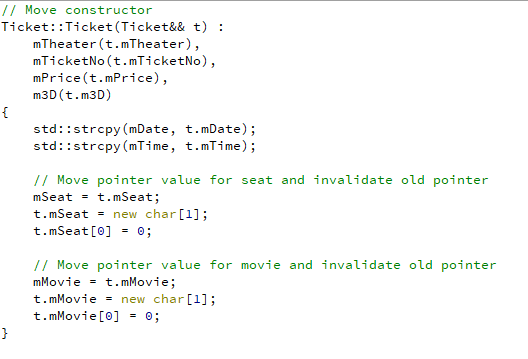
**<chrono>:**

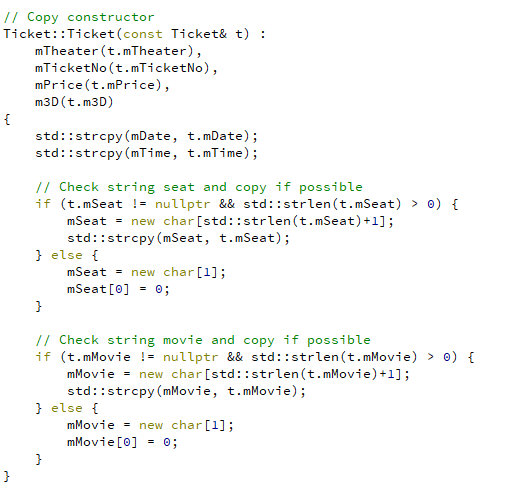
Kan bruges til at måle tid:

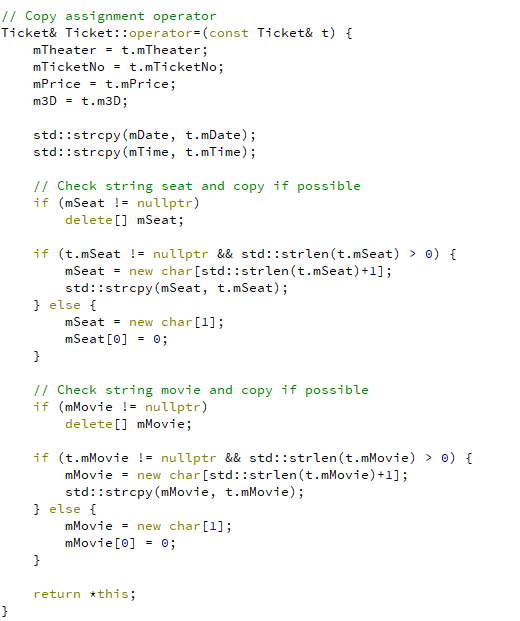


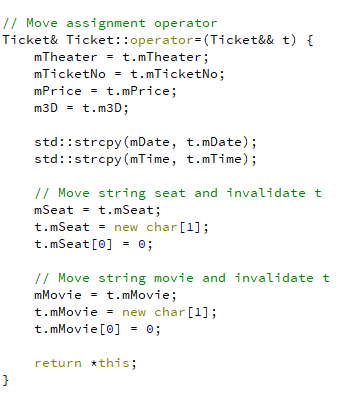
**Get-metoder, set-metoder, move-constructors og move-assignment samt copy-constructors og copy-assignment med flere datatyper.**

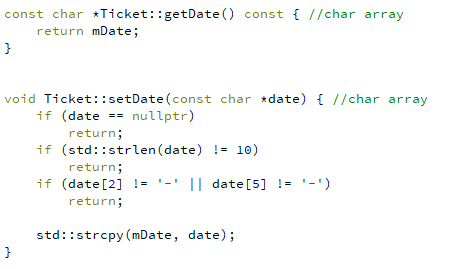


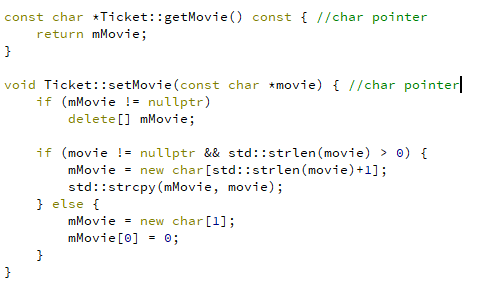






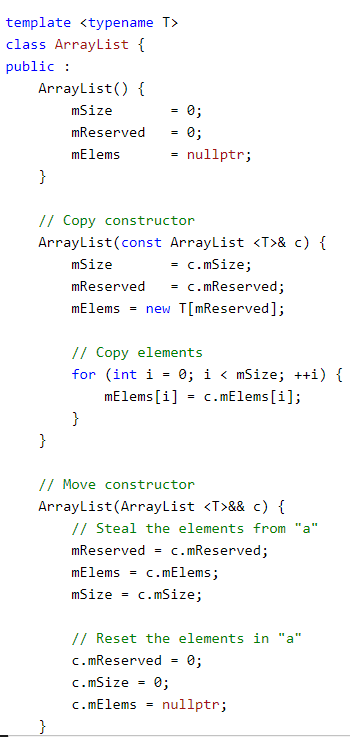


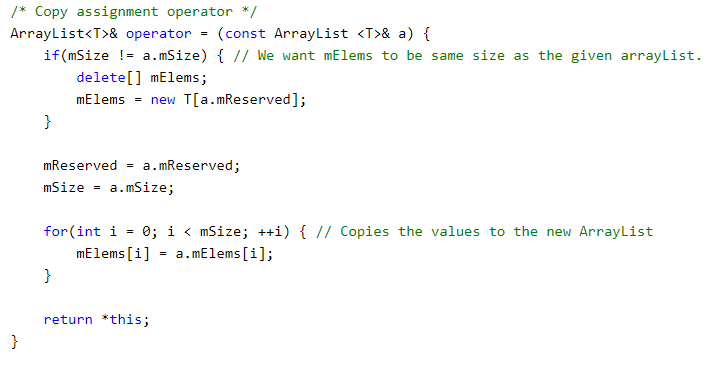




ENDNU et godt eksempel:

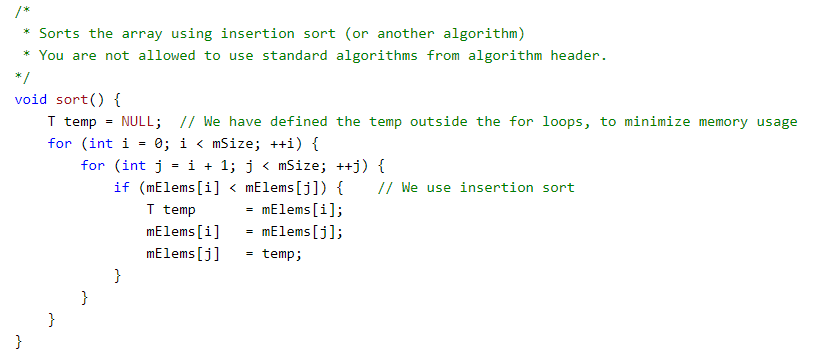
Her er mSize = int, mReserved = int og mElems = pointer til array af typen T (template klasse).



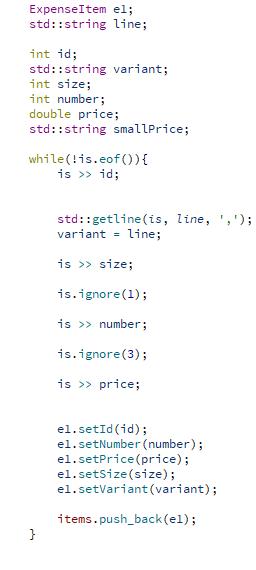




Sorteringsalgoritme:



**Læse filer med std::istream**

 **Format:**

