

# Polymorfi og abstrakte klasser

---

Forelesning 7,  
Effektiv kode med C og C++, vår 2015  
Alfred Bratterud

Pensum: Kap.14



# Agenda:

- \* Copy- og move-konstruktører
- \* Arv så langt
- \* Polymorfi
- \* Abstrakte klasser



# Copy og assign

- \* Anta `myclass a{};`
- \* Hva skjer her: `myclass b(a);`
  - \* Vi \*mener\* at vi vil lage en instans `b`, som er identisk som `a`
  - \* Hvilken konstruktør kaller vi?
  - \* `myclass(const myclass& y)` - evt. `myclass(myclass y)`: copy-constructor!
- \* Eller her: `a=b;`
  - \* Vi \*mener\* at innholdet i `b` skal \*kopieres inn\* i `a`
  - \* Hvilken operator brukes?
  - \* `myclass& operator=(myclass& y)`
  - \* `type operatorsymbol (parametre) { ... }`
- \* En "copy constructor" er en constructor som tar sin egen type som argument
  - \* Hva hvis `myclass` inneholder pekere? Pekere betyr alltid trøbbel...
  - \* Alle klasser får en «defualt» copy-constructor, men den gjør \*ikke\* deep copy.
  - \* Det må du lage selv! (...hvorfor?)



# Copy og assign

- \* 3-regelen - «Trenger du en trenger du alle»:
  - \* **Destructor**
    - \* `~myclass()`
  - \* **Copy constructor**
    - \* `myclass(const myclass&);`
  - \* **Copy assignment operator**
    - \* `myclass& operator=(const myclass&)`
- \* Kan vi klare oss med «default»?
  - \* Veldig ofte
  - \* Men ofte ikke, når vi har pekere som medlemmer :-)



# C++11:

## «Move semantics»

- \* C++11 introduserer to til:

- \* Move constructor

- \* `myclass(myclass&& c)`  
`{ /* ...flytt alle data*/ }`

- \* Move assignment operator

- \* `myclass& operator=(const myclass&& c)`  
`{ /* ...flytt alle data*/ }`

- \* Fordi «flytting» kan være billigere enn kopiering

- \* ?!

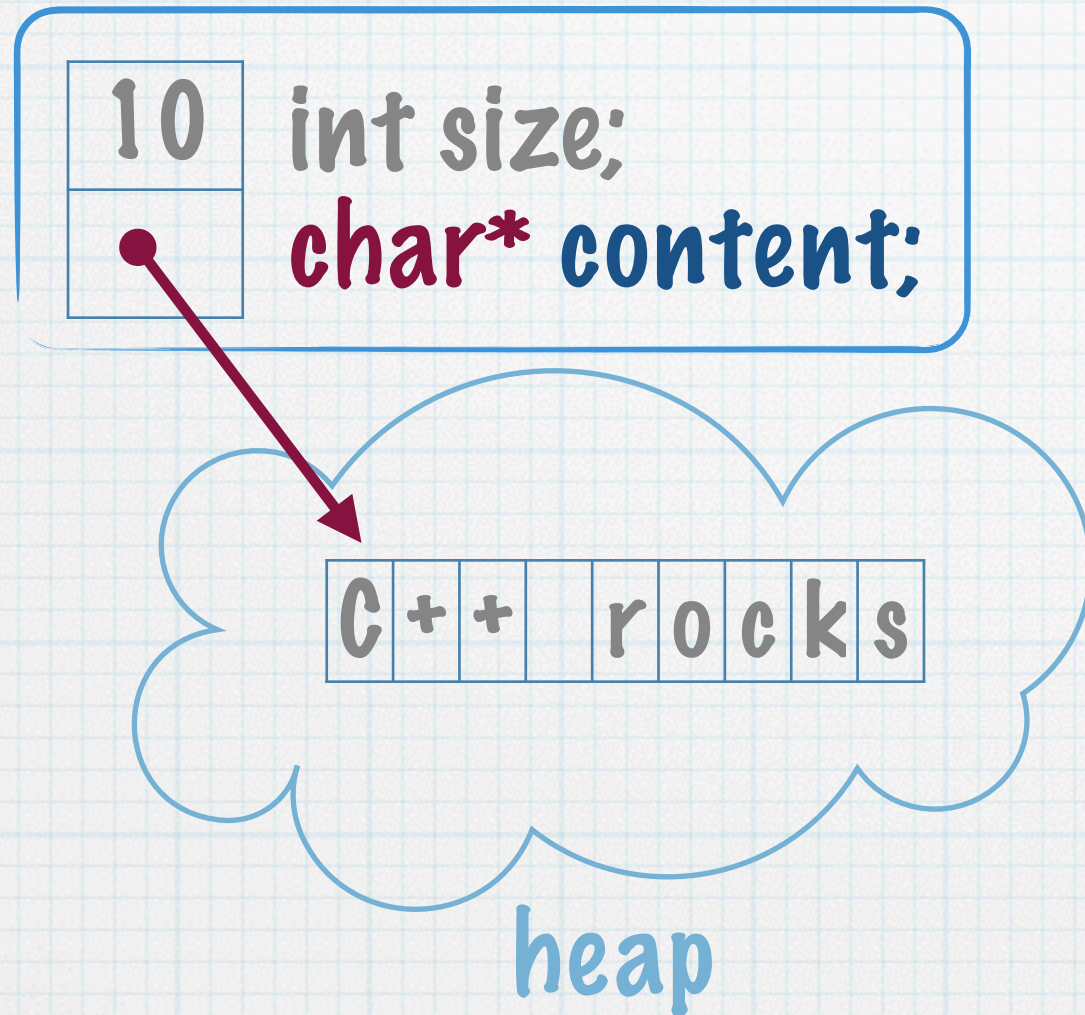
- \* Igjen - tenk pekere (som forårsaker alle problemer i C/C++)



# Copy:

MyString a

MyString b(a) / MyString b=a



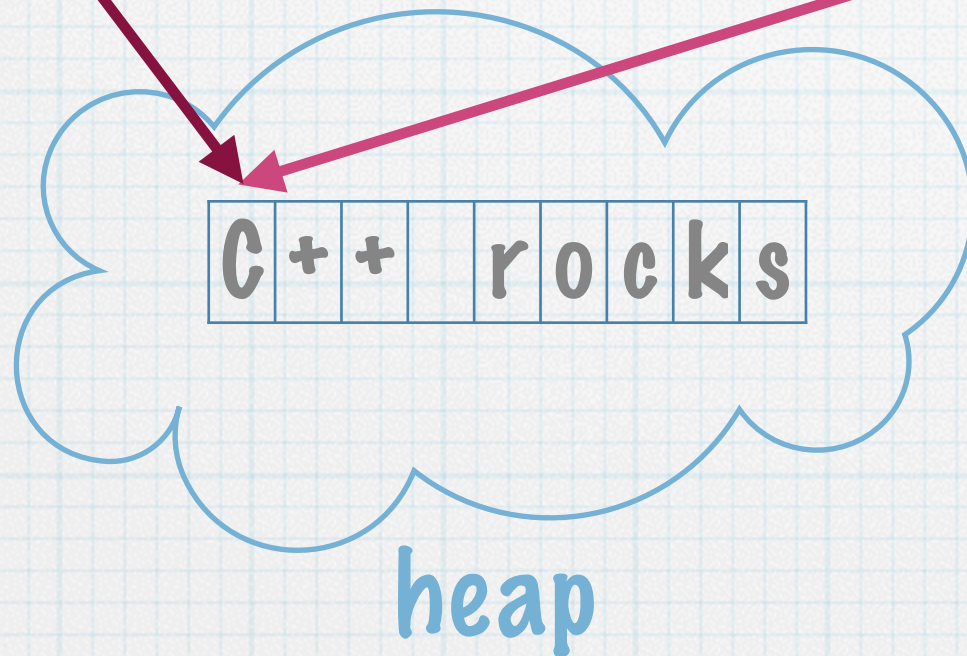
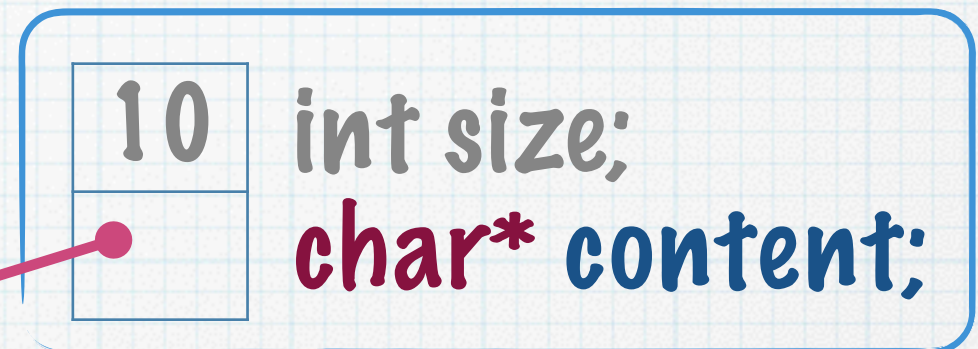
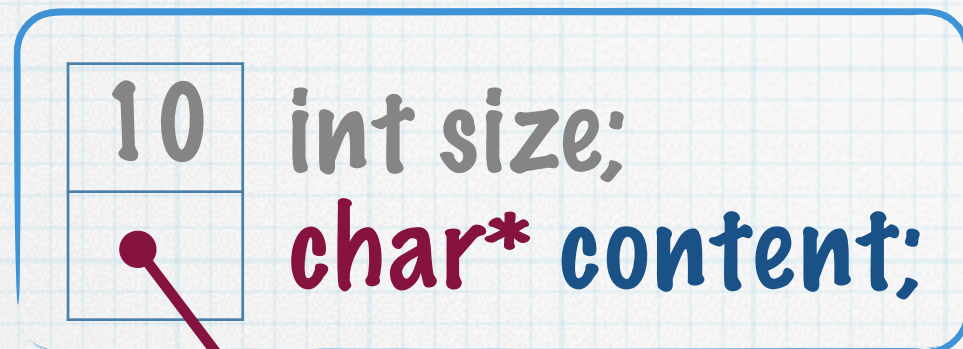
?



# Copy:

MyString a

MyString b(a) / MyString b=a



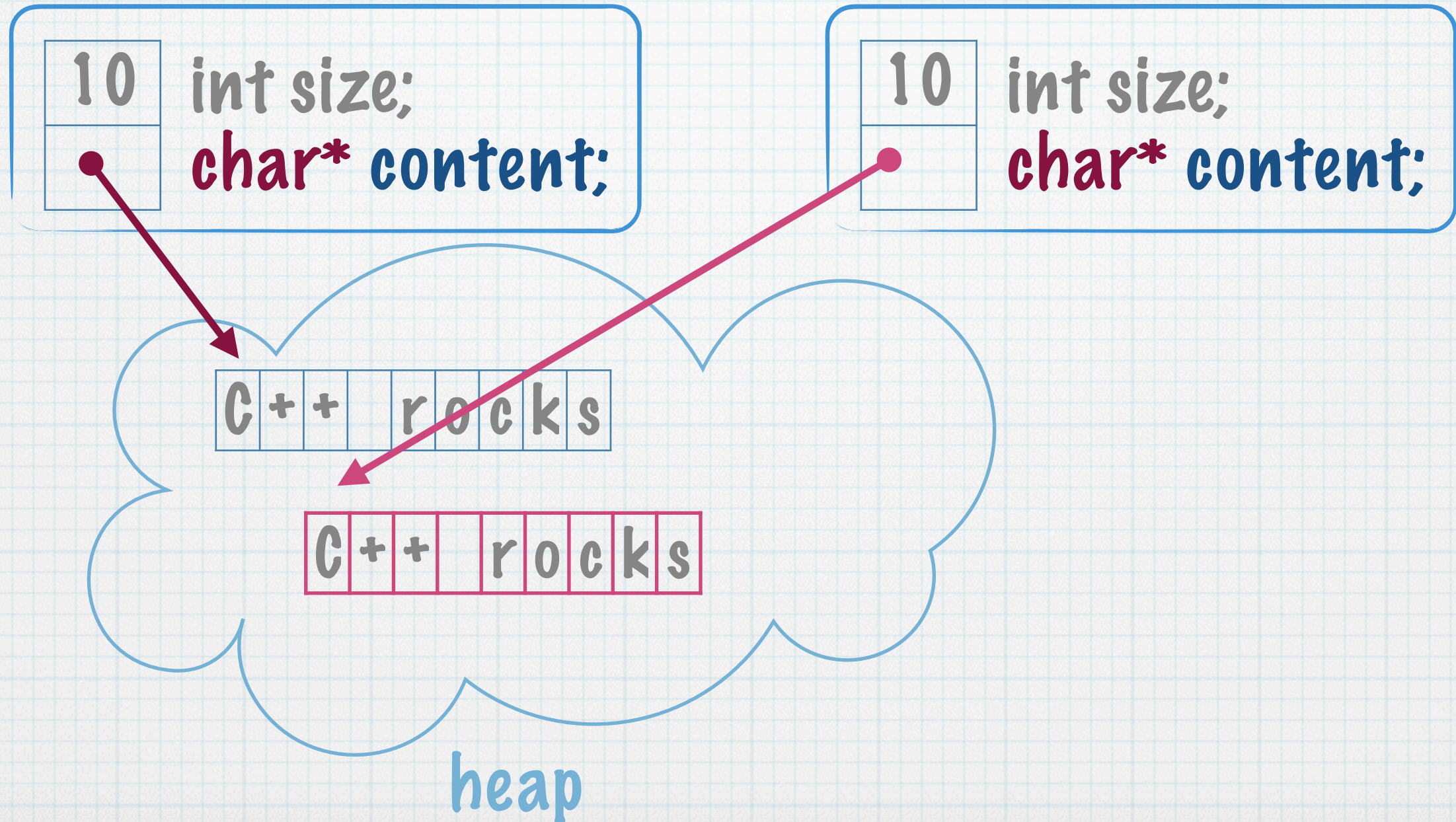
Default for  
pekere!



# Copy:

MyString a

MyString b(a) / MyString b=a

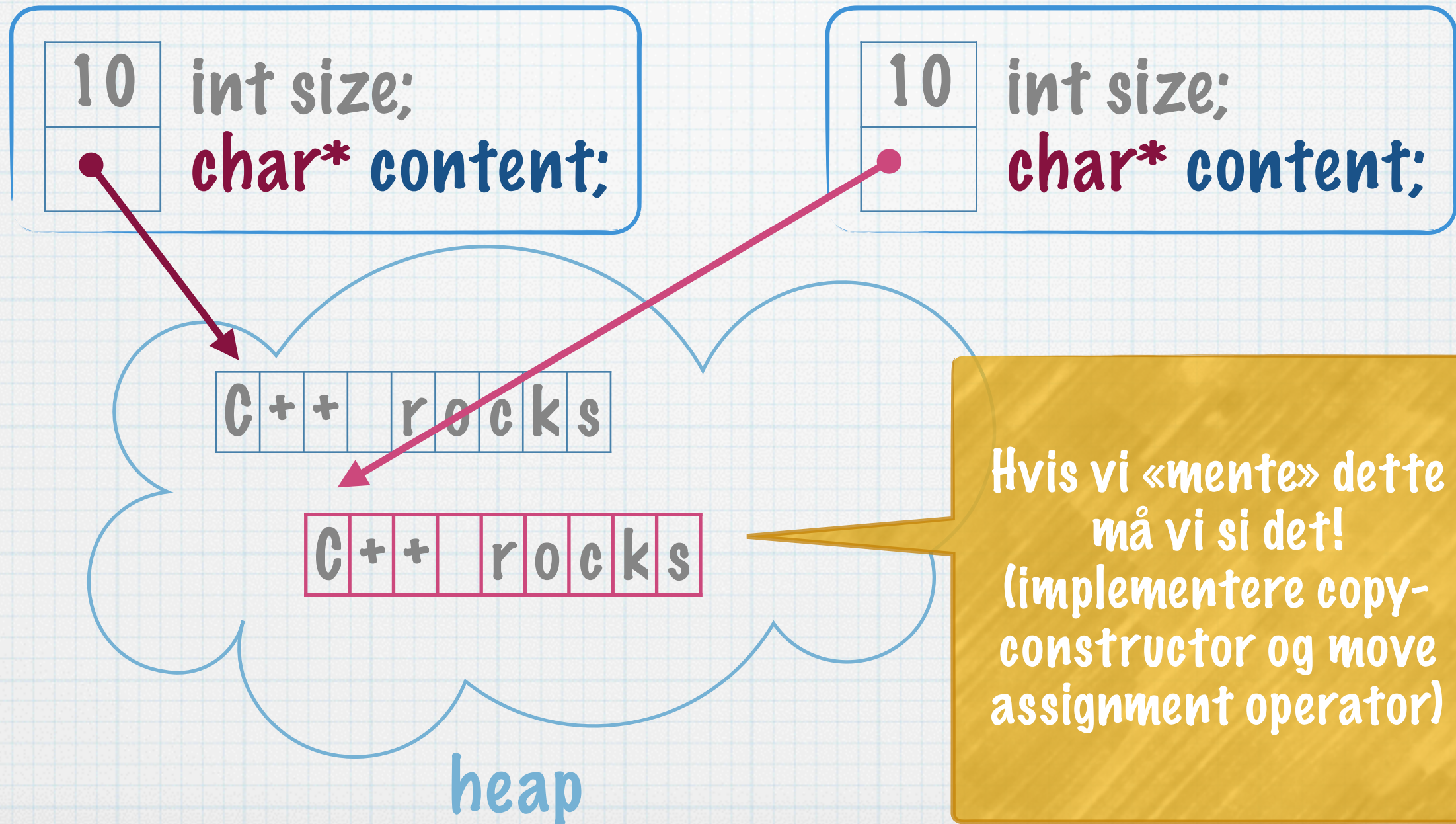




# Copy:

MyString a

MyString b(a) / MyString b=a

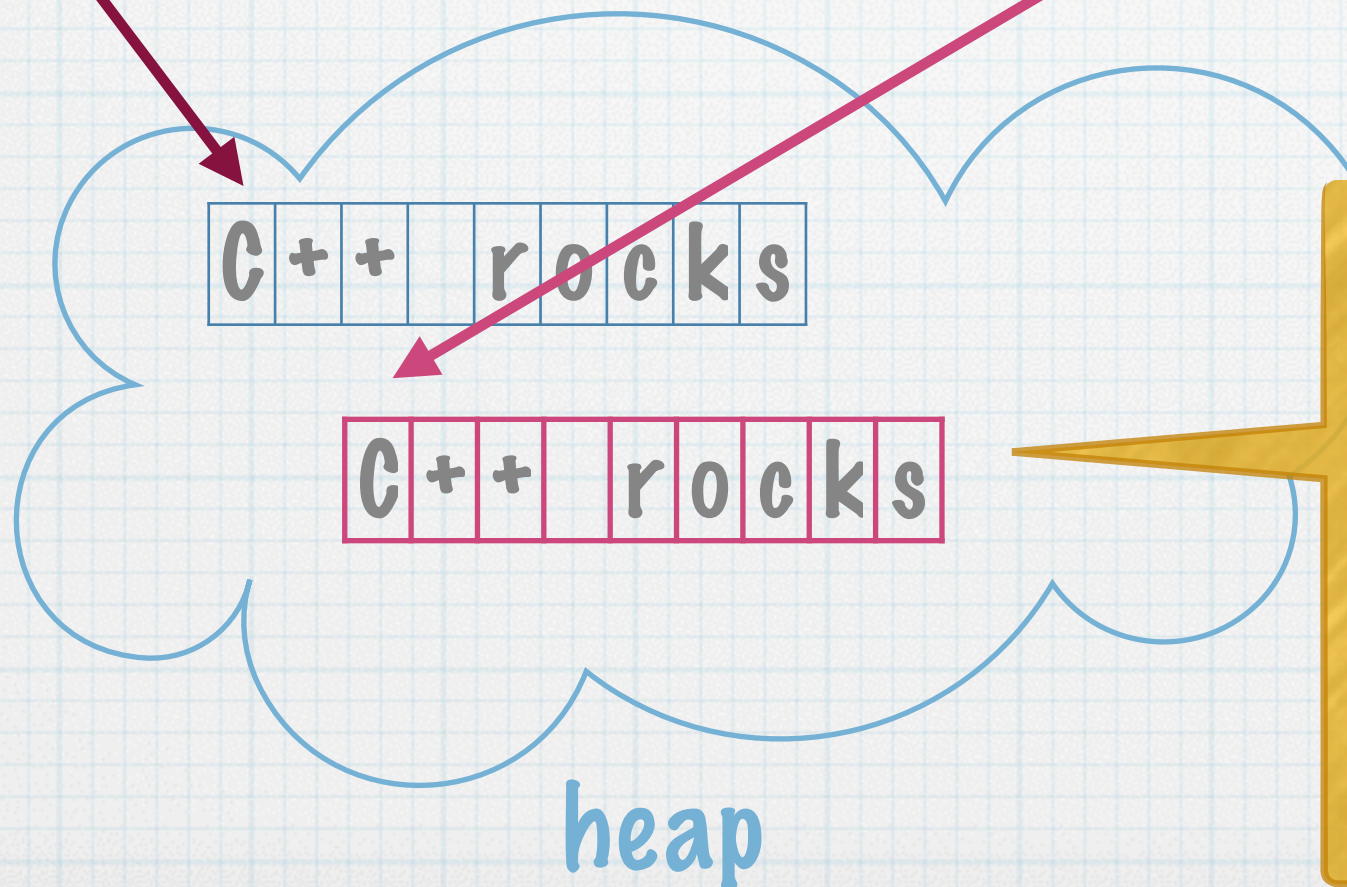
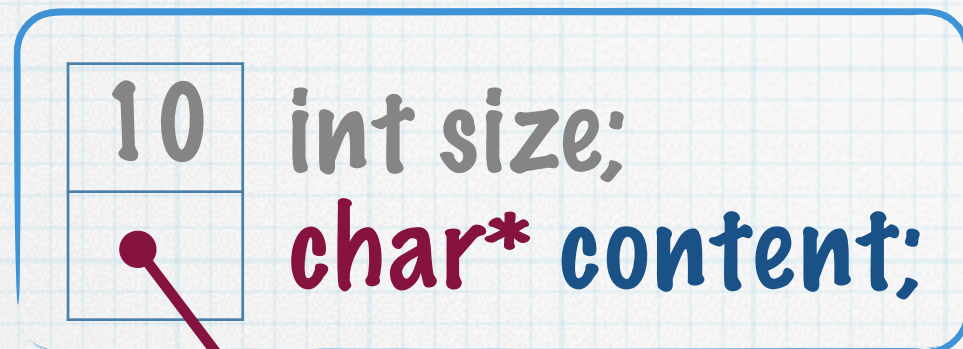




# Copy:

MyString a

MyString b(a) / MyString b=a



Vil vi gjøre denne  
kopieringen to steder i  
koden?

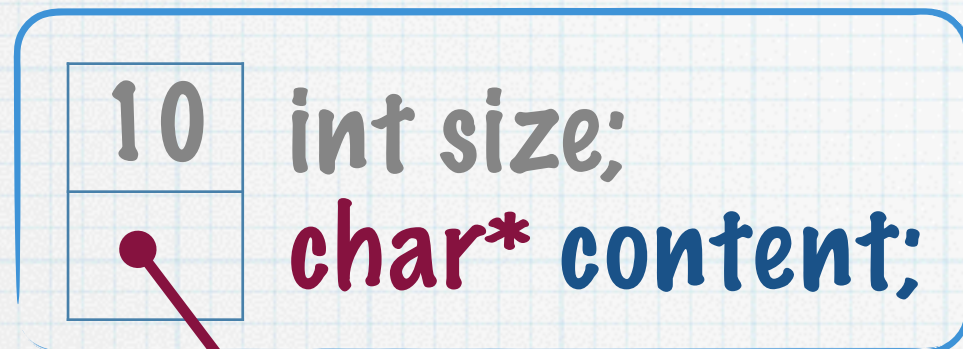
NEI!

Lag en «copy»-funksjon



# Move:

**MyString a**



C + +   r o c k s

a's data på heap

**MyString b = reverse(a);**

Anta «by-value:»  
**MyString reverse(MyString s)**

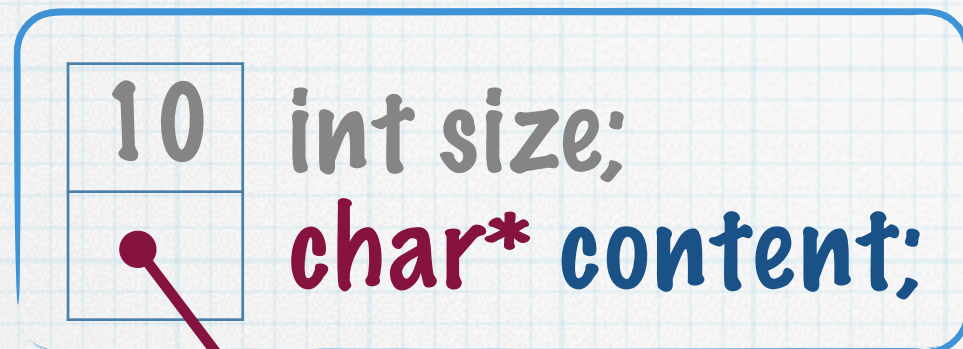
og at vi har copy-constructor og  
copy-assignment operator som  
gjør «deep copy»



# Move:

MyString a

MyString b = reverse(a);



C + +   r o c k s

reverse(a){ ... }

int size;  
char\* content;



C + +   r o c k s

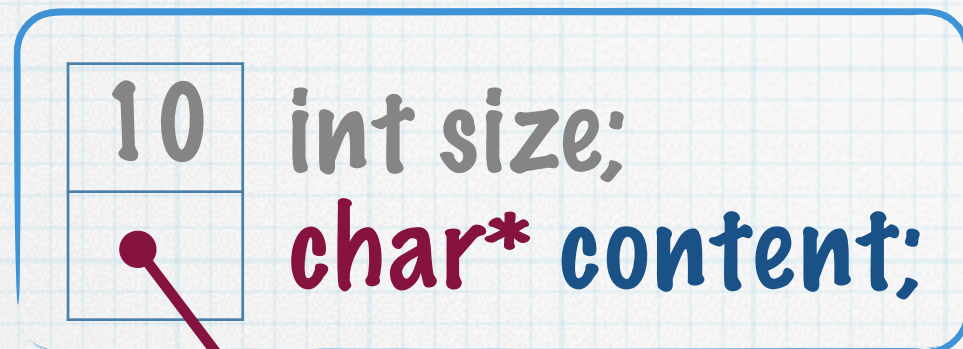
reverse - kopi av a's  
data på heap



# Move:

MyString a

MyString b = reverse(a);



C + + r o c k s

reverse(a){ ... }

int size;  
char\* content;



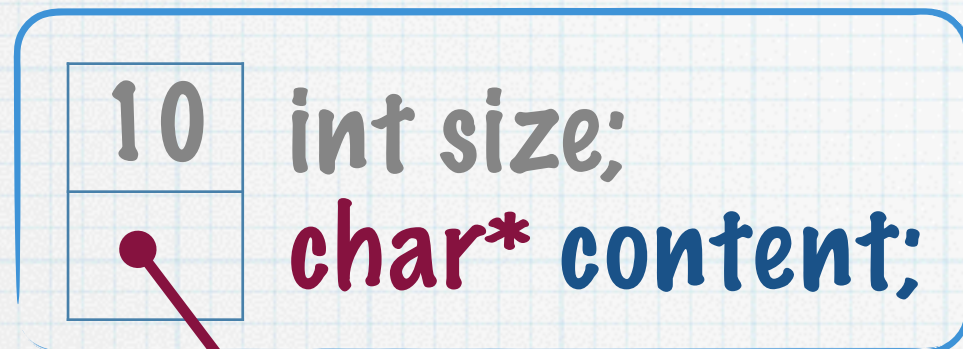
C + + s u c k s

reverse - kopi av a's  
data på heap



# Move:

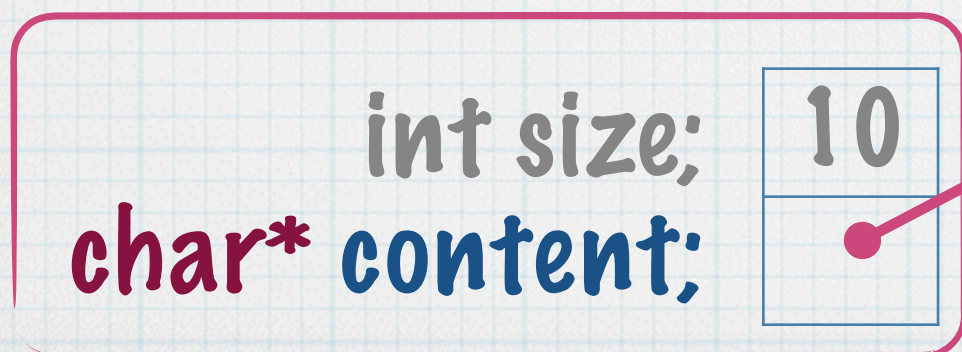
MyString a



MyString b = reverse(a);



reverse(a){ ... }



reverse - kopi av a's  
data på heap



# Move:

**MyString a**

Før C++11 måtte vi bruke copy/assign til dette. Trenger vi tre kopier?

**a** → 

C	+	+		r	o	c	k	s
---	---	---	--	---	---	---	---	---

**reverse(a){ ... }**

int size;	10
char* content;	•

**MyString b = reverse(a);**

int size;	10
char* content;	•

C	+	+		s	u	c	k	s
---	---	---	--	---	---	---	---	---

C	+	+		s	u	c	k	s
---	---	---	--	---	---	---	---	---

reverse - kopi av a's data på heap



# Move:

**MyString a**

Før C++11 måtte vi bruke  
copy/assign til dette.  
Trenger vi tre kopier?

**a** → 

C	+	+		r	o	c	k	s
---	---	---	--	---	---	---	---	---

reverse sin kopi på stack går  
ut av skop - og poppes

**MyString b = reverse(a);**

int size;	10
<b>char* content;</b>	●

→ 

C	+	+		s	u	c	k	s
---	---	---	--	---	---	---	---	---

C	+	+		s	u	c	k	s
---	---	---	--	---	---	---	---	---

reverse - kopi av a's  
data på heap



# Move:

**MyString a**

Før C++11 måtte vi bruke  
copy/assign til dette.  
Trenger vi tre kopier?

**a** → 

C	+	+		r	o	c	k	s
---	---	---	--	---	---	---	---	---

**MyString b = reverse(a);**

int size;	10
<b>char* content;</b>	•

↓

C	+	+		s	u	c	k	s
---	---	---	--	---	---	---	---	---

C	+	+		s	u	c	k	s
---	---	---	--	---	---	---	---	---



reverse - kopi av a's  
data på heap



# Move:

**MyString a**

Før C++11 måtte vi bruke  
copy/assign til dette.  
Trenger vi tre kopier?

**a** → 

C	+	+		r	o	c	k	s
---	---	---	--	---	---	---	---	---

**MyString b = reverse(a);**

int size;	10
<b>char* content;</b>	●

↙

C	+	+		s	u	c	k	s
---	---	---	--	---	---	---	---	---

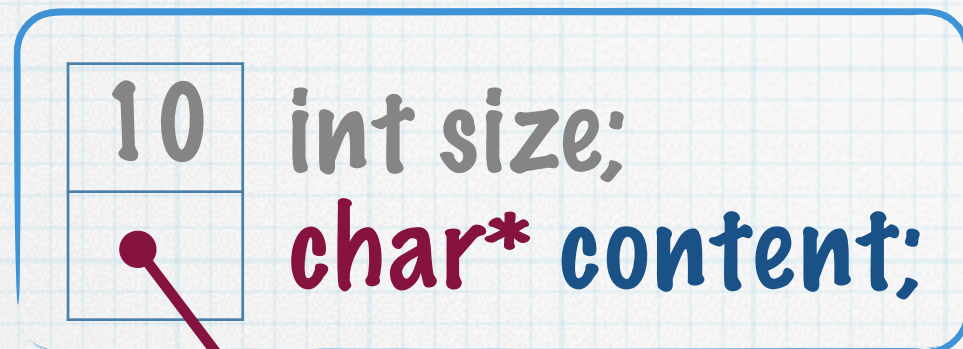
MyStrings destructor  
rydder opp  
(hvis den er laget riktig)  
etter reverse på heap.



# Move C++ 11

MyString a

MyString b = reverse(a);



C + + r o c k s

reverse(a){ ... }



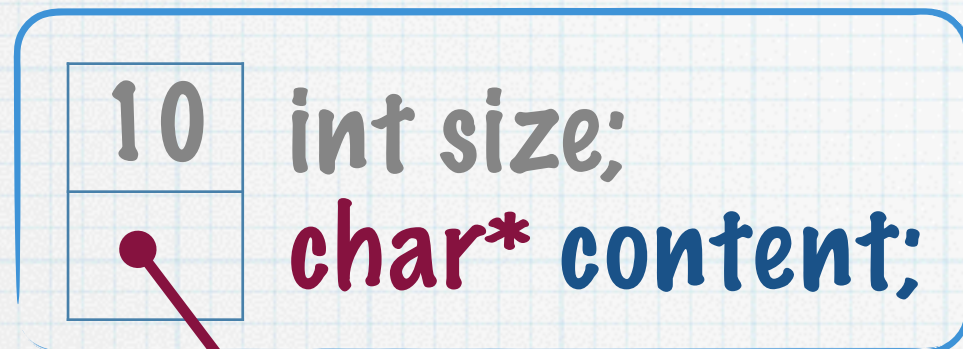
C + + s u c k s

reverse - kopi av a's  
data på heap



# Move C++ 11

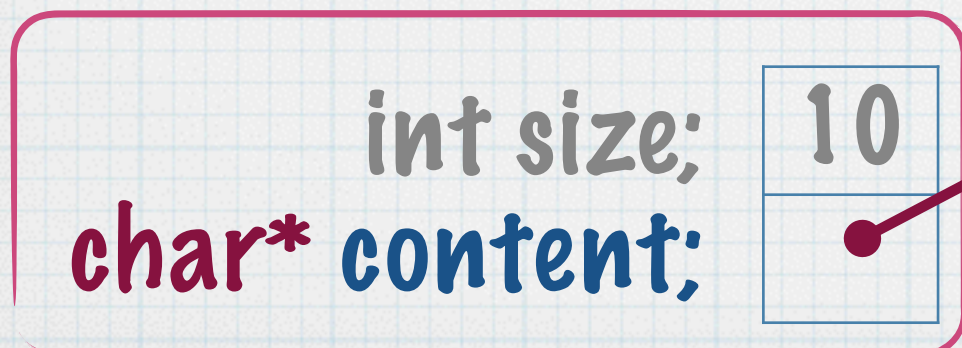
MyString a



MyString b = reverse(a);



reverse(a){ ... }

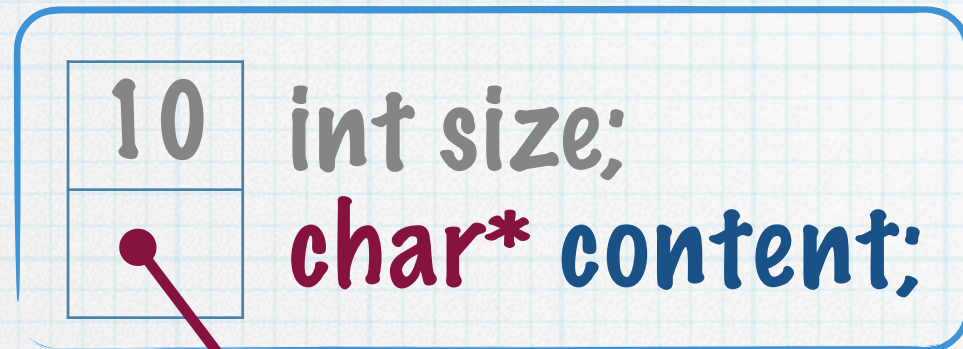


reverse - kopi av a's  
data på heap



# Move C++11

MyString a



C + + r o c k s

reverse(a){ ... }



MyString b = reverse(a);



C + + s u c k s

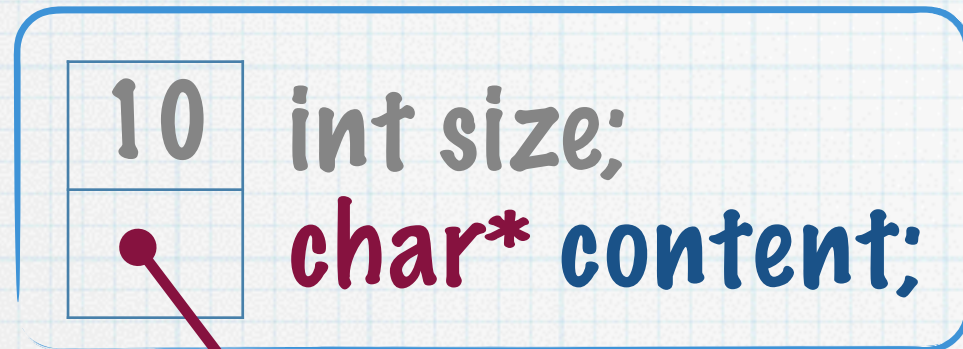
Move-konstruktor  
«Stjeler» dataene!»



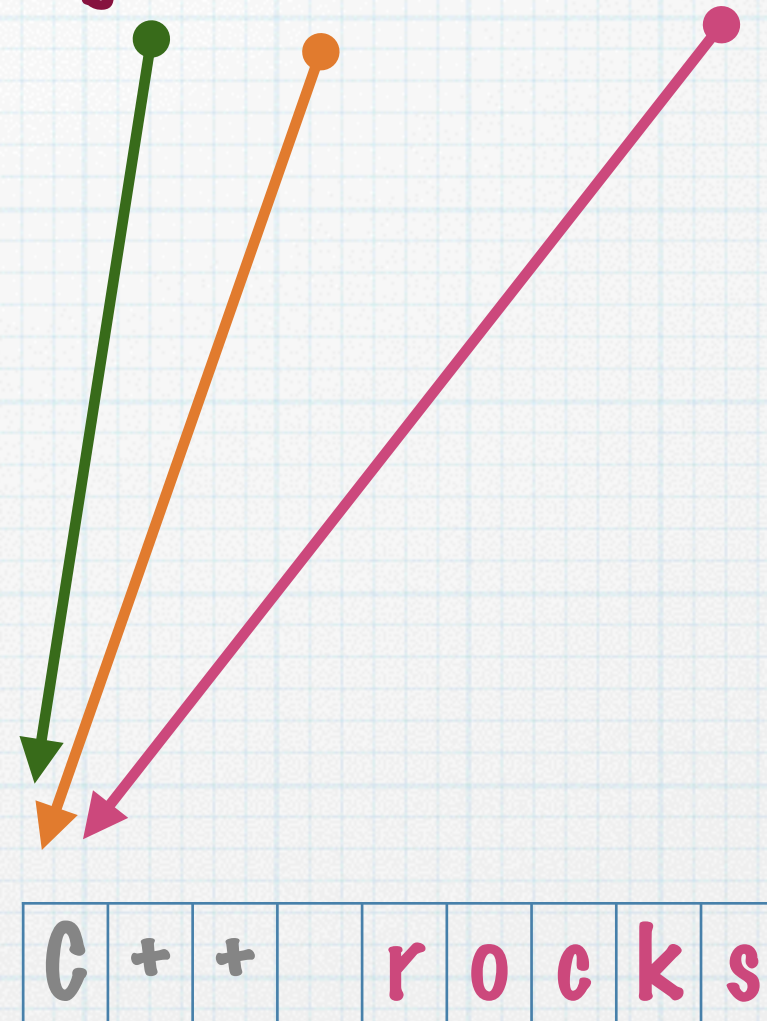
# Move C++ 11

MyString a

MyString b = reverse(reverse(a));



C + + r o c k s





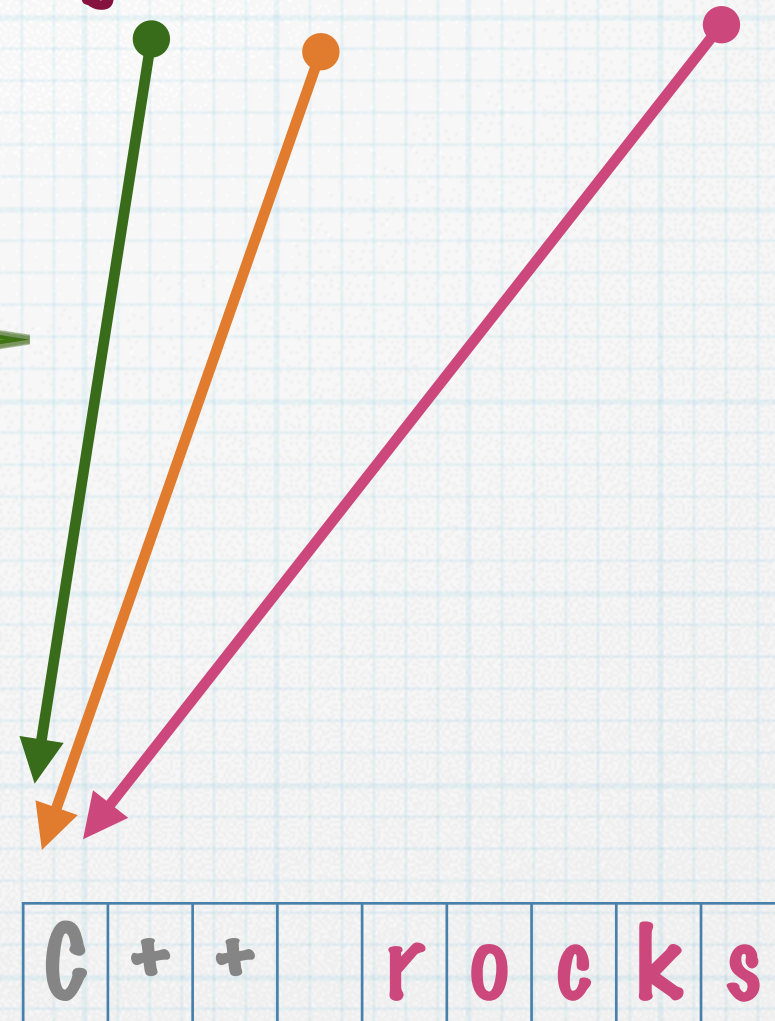
# Move C++ 11

MyString a

MyString b = reverse(reverse(a));

Og det skalerer!  
Nøstede kall til  
reverse kan bruke  
samme løsning.

Her: sparer 2  
kopieringer!





# C++11:

## «Move semantics»

- \* «Move-konstruktør» / «Move assignment» vil bare bli kalt av kompilator, når den «vet» at «kilden» skal ut av skop
- \* Eller, når variabel er «på venstre side» av en «rvalue» av samme type
- \* «rvalue»: en midlertidig variabel uten navn:  $(a+b)$ , `resultOf(x)` etc.
- \* «lvalue»: en variabel \*med\* navn - noe som kan stå på «left side» i en tilordning.



# Å HUSKE: 5-regelen

- \* «Trenger du en trenger du alle»:

- \* **Destructor**

  - \* `~myclass()`

- \* **Copy constructor**

  - \* `myclass(const myclass&);`

- \* **Copy assignment operator**

  - \* `myclass& operator=(const myclass&)`

- \* **Move constructor**

  - \* `myclass(myclass&& c)`

- \* **Move assignment operator**

  - \* `myclass& operator=(const myclass&& c)`



# Å HUSKE: 5-regelen

- \* «Trenger du en trenger du alle»:

- \* **Destructor**

- \* `~myclass()`



Rydd opp

- \* **Copy constructor**

- \* `myclass(const myclass&);`

- \* **Copy assignment operator**

- \* `myclass& operator=(const myclass&)`

- \* **Move constructor**

- \* `myclass(myclass&& c)`

- \* **Move assignment operator**

- \* `myclass& operator=(const myclass&& c)`



# Å HUSKE: 5-regelen

- \* «Trenger du en trenger du alle»:

- \* **Destructor**

- \* `~myclass()`

- \* **Copy constructor**

- \* `myclass(const myclass&);`

- \* **Copy assignment operator**

- \* `myclass& operator=(const myclass&)`

- \* **Move constructor**

- \* `myclass(myclass&& c)`

- \* **Move assignment operator**

- \* `myclass& operator=(const myclass&& c)`



Deep Copy



# Å HUSKE: 5-regelen

- \* «Trenger du en trenger du alle»:

- \* **Destructor**

  - \* `~myclass()`

- \* **Copy constructor**

  - \* `myclass(const myclass&);`

- \* **Copy assignment operator**

  - \* `myclass& operator=(const myclass&)`

- \* **Move constructor**

  - \* `myclass(myclass&& c)`

- \* **Move assignment operator**

  - \* `myclass& operator=(const myclass&)`



Steal!



# Demo:

---

`copy_move.cpp`

Eksempler:

<http://www.cplusplus.com/articles/y8hv0pDG/>  
<http://en.cppreference.com/w/cpp/language/>



# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
- \* Multippel arv gir mulighet for "Mixin"
- \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
- \* Kan vi legge flere typer objekter i samme container?
- \* Kan vi få ulike varianter av en baseklasse til å oppføre seg forskjellig- selv om vi betrakter dem som "base-type"?



# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
- \* **Ja - med abstrakte klasser og multippel arv**
- \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
- \* Kan vi legge flere typer objekter i samme container?
- \* Kan vi få ulike varianter av en baseklasse til å oppføre seg forskjellig- selv om vi betrakter dem som "base-type"?



# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
- \* **Ja - med abstrakte klasser og multippel arv**
- \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
  - Ja - hvis de har felles baseklasse**
- \* Kan vi legge flere typer objekter i samme container?
- \* Kan vi få ulike varianter av en baseklasse til å oppføre seg forskjellig- selv om vi betrakter dem som "base-type"?



# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
- \* **Ja - med abstrakte klasser og multippel arv**
- \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
  - Ja - hvis de har felles baseklasse**
- \* Kan vi legge flere typer objekter i samme container?
  - Ja - med polymorfi**
- \* Kan vi få ulike varianter av en baseklasse til å opptøre seg forskjellig- selv om vi betrakter dem som "base-type"?

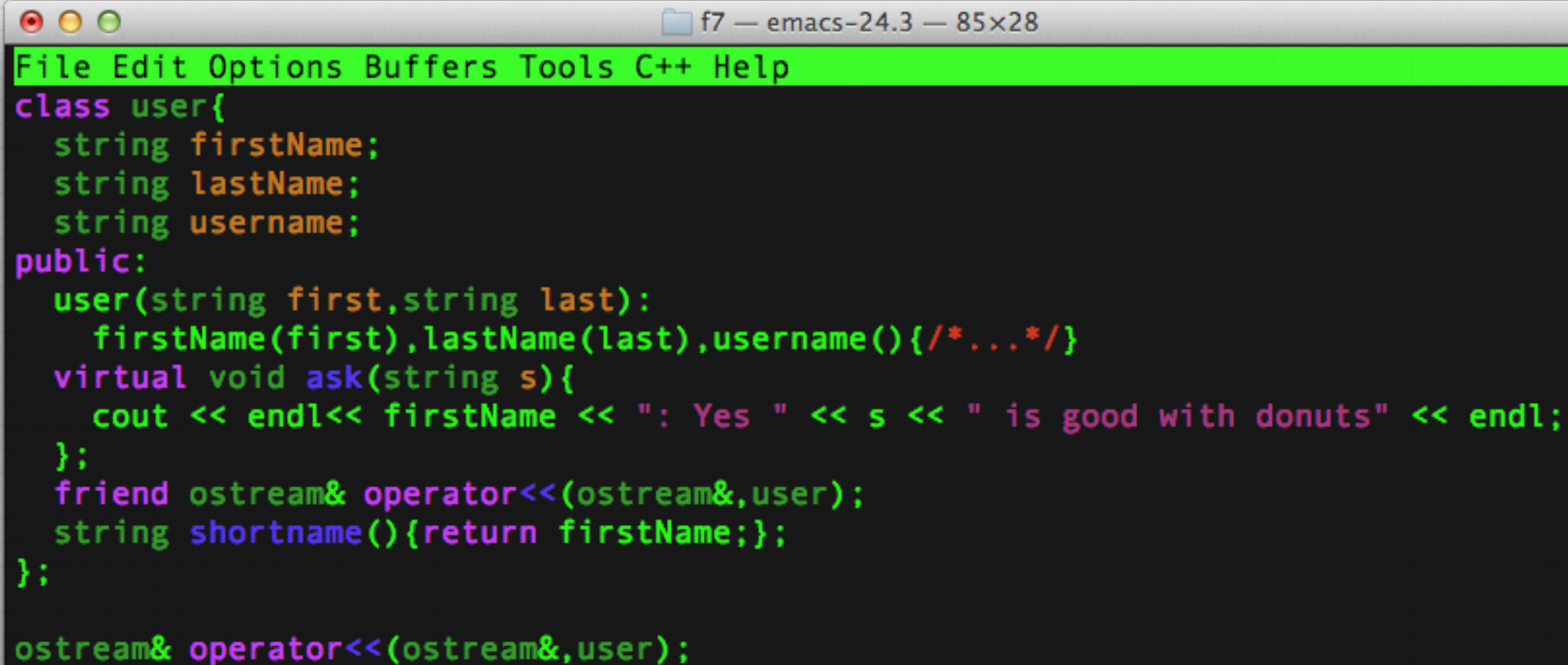


# Polymorfi

- \* “Poly” betyr mange, “morf” betyr “form”
- \* Polymorfi er når et kall på en funksjon **f** i en instans **B**, sender kallet nedover i “arv”, til subklassen **S**.
- \* For at det skal skje må **f** være deklarerert **virtual** og definert både i **B** og i **S**. Definisjonen i subklassen er da en “**override**” (kan/bør spesifiseres i C++ 11)
- \* Hvis man ikke bruker **virtual** kan **B** og **S** fint ha funksjonen **f** men den er da ikke “overridet” og funker kun når man har en peker av subtype.



# Polymorfi



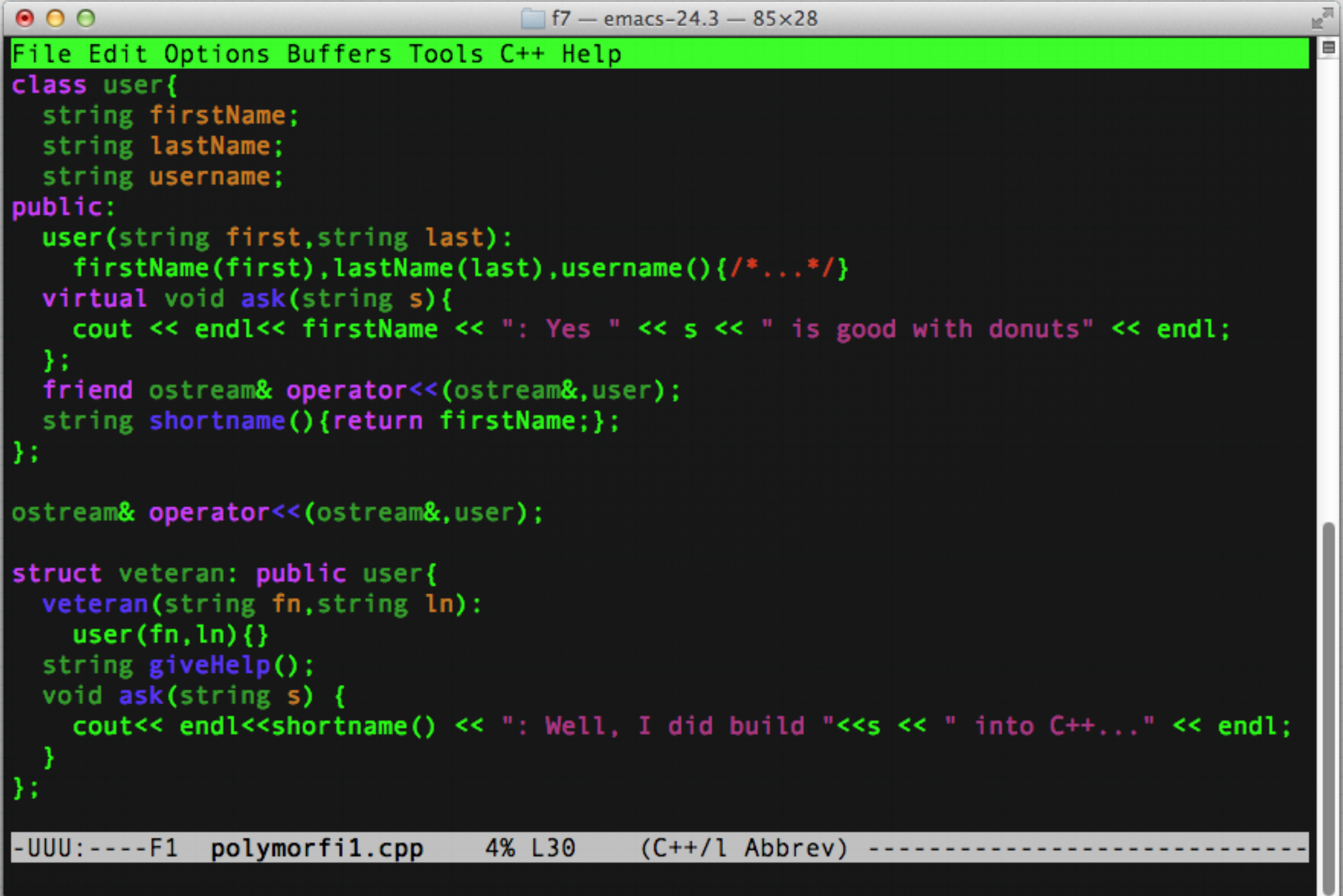
The image shows a screenshot of an Emacs editor window. The title bar at the top reads "f7 — emacs-24.3 — 85x28". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "C++", and "Help". The main text area contains C++ code for a class named 'user'. The code is color-coded: keywords like 'class', 'public', 'virtual', 'friend', and 'return' are in purple; identifiers like 'firstName', 'lastName', 'username', 'first', 'last', 's', and 'shortname' are in orange; and string literals and comments are in green. The code defines a class 'user' with three member variables, a constructor, a virtual 'ask' method, and a 'shortname' method. It also shows a friend function for operator overloading and a line of code for another operator overload.

```
File Edit Options Buffers Tools C++ Help
class user{
    string firstName;
    string lastName;
    string username;
public:
    user(string first,string last):
        firstName(first),lastName(last),username() { /*...*/ }
    virtual void ask(string s){
        cout << endl<< firstName << ": Yes " << s << " is good with donuts" << endl;
    };
    friend ostream& operator<<(ostream&,user);
    string shortname(){return firstName;};
};

ostream& operator<<(ostream&,user);
```



# Polymorfi



The image shows a screenshot of an Emacs editor window. The title bar at the top reads "f7 — emacs-24.3 — 85x28". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "C++", and "Help". The main editing area contains C++ code for a polymorphic class hierarchy. The code defines a base class "user" and a derived class "veteran". The "user" class has attributes "firstName", "lastName", and "username", and methods "user()", "ask()", "shortname()", and an overloaded "operator<<". The "veteran" struct inherits from "user" and overrides the "ask()" method. The status bar at the bottom shows "-UUU:----F1", the filename "polymorfi1.cpp", the cursor position "4% L30", and the mode "(C++/l Abbrev)".

```
File Edit Options Buffers Tools C++ Help
class user{
    string firstName;
    string lastName;
    string username;
public:
    user(string first,string last):
        firstName(first),lastName(last),username() { /*...*/ }
    virtual void ask(string s){
        cout << endl<< firstName << ": Yes " << s << " is good with donuts" << endl;
    };
    friend ostream& operator<<(ostream&,user);
    string shortname(){return firstName;};
};

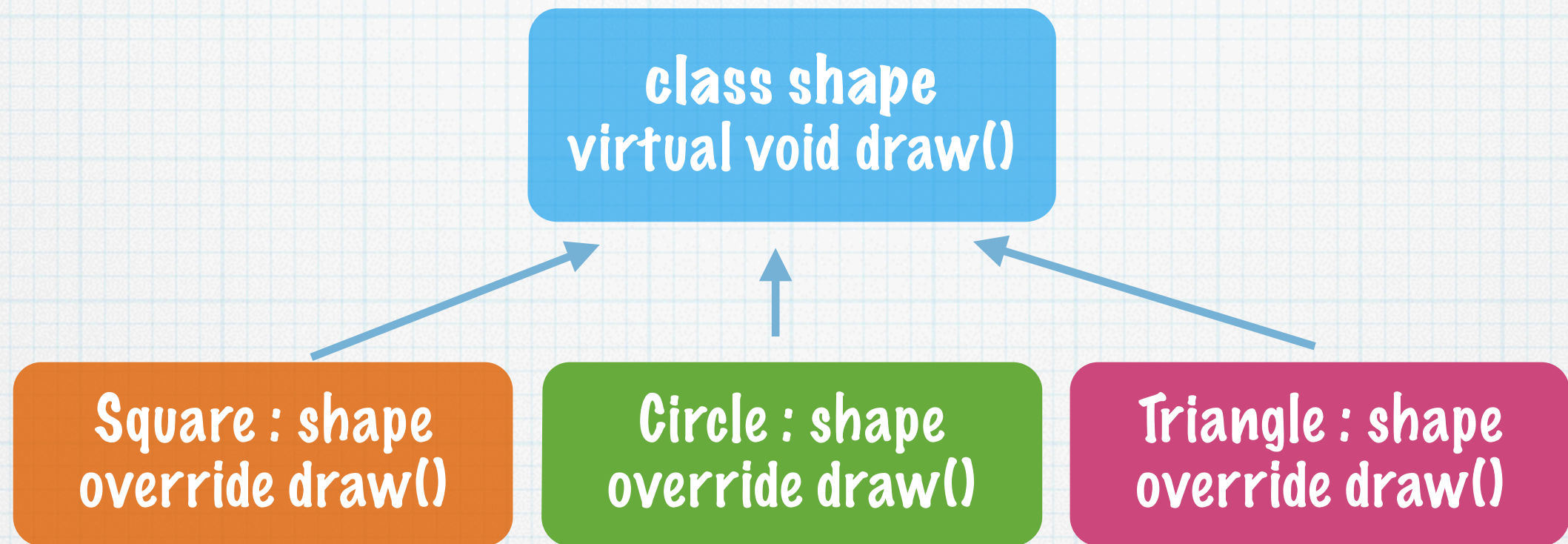
ostream& operator<<(ostream&,user);

struct veteran: public user{
    veteran(string fn,string ln):
        user(fn,ln){}
    string giveHelp();
    void ask(string s) {
        cout<< endl<<shortname() << ": Well, I did build "<<s << " into C++..." << endl;
    }
};

-UUU:----F1  polymorfi1.cpp    4% L30    (C++/l Abbrev) -----
```



# Polymorfi





# Polymorfi

shape

Square

Circle

Triangle

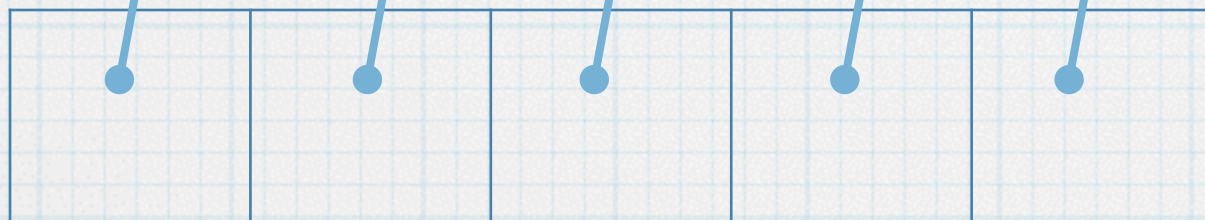
shape

shape

shape

shape

shape



`Vector<shape*> shapes`



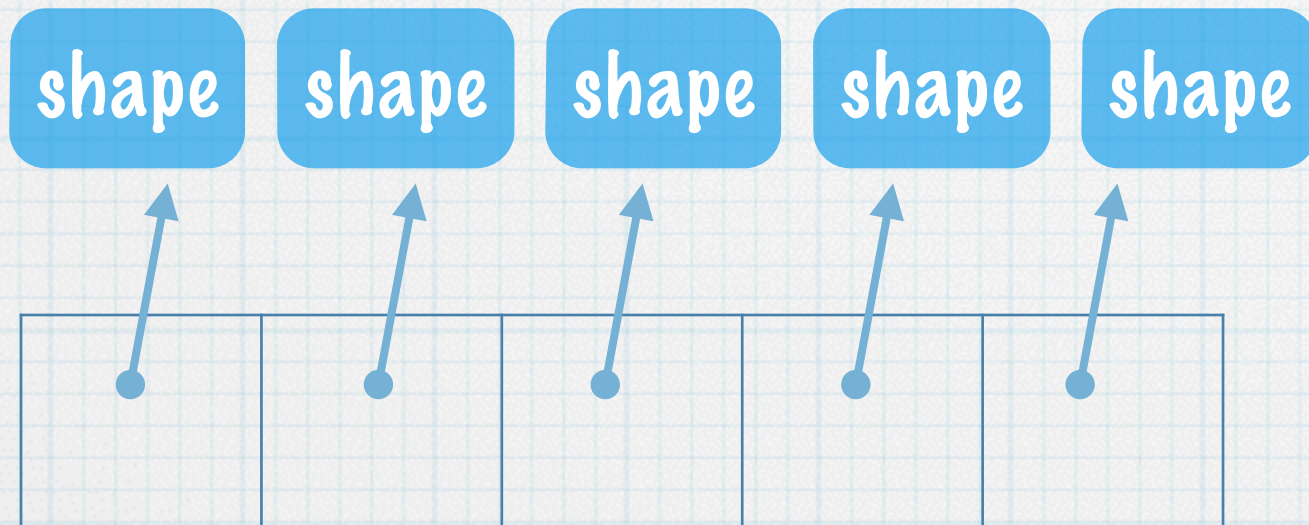
# Polymorfi

shape

Square

Circle

Triangle

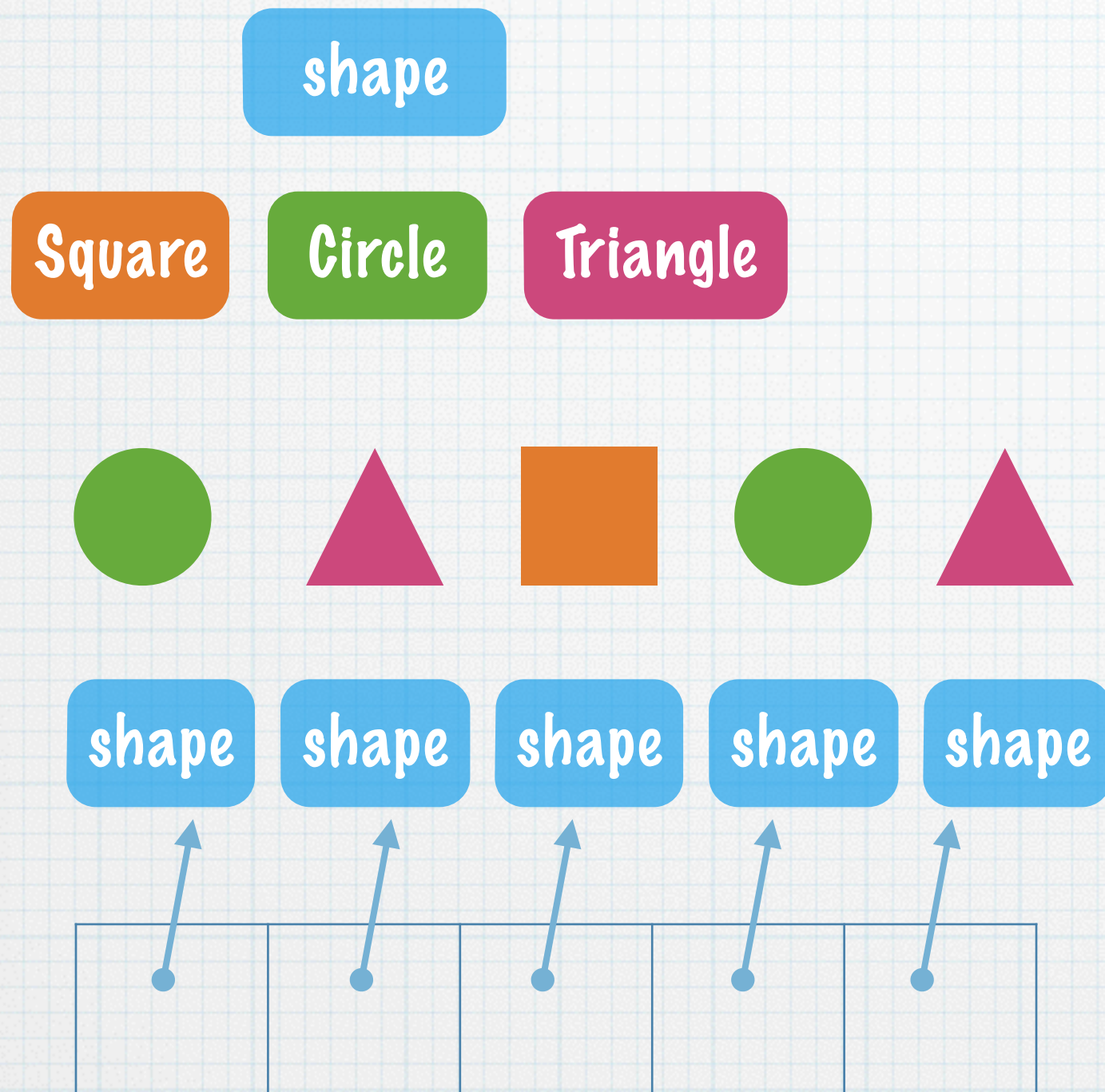


`Vector<shape*> shapes`

```
for ( auto s : shapes )  
    s->draw();
```



# Polymorfi

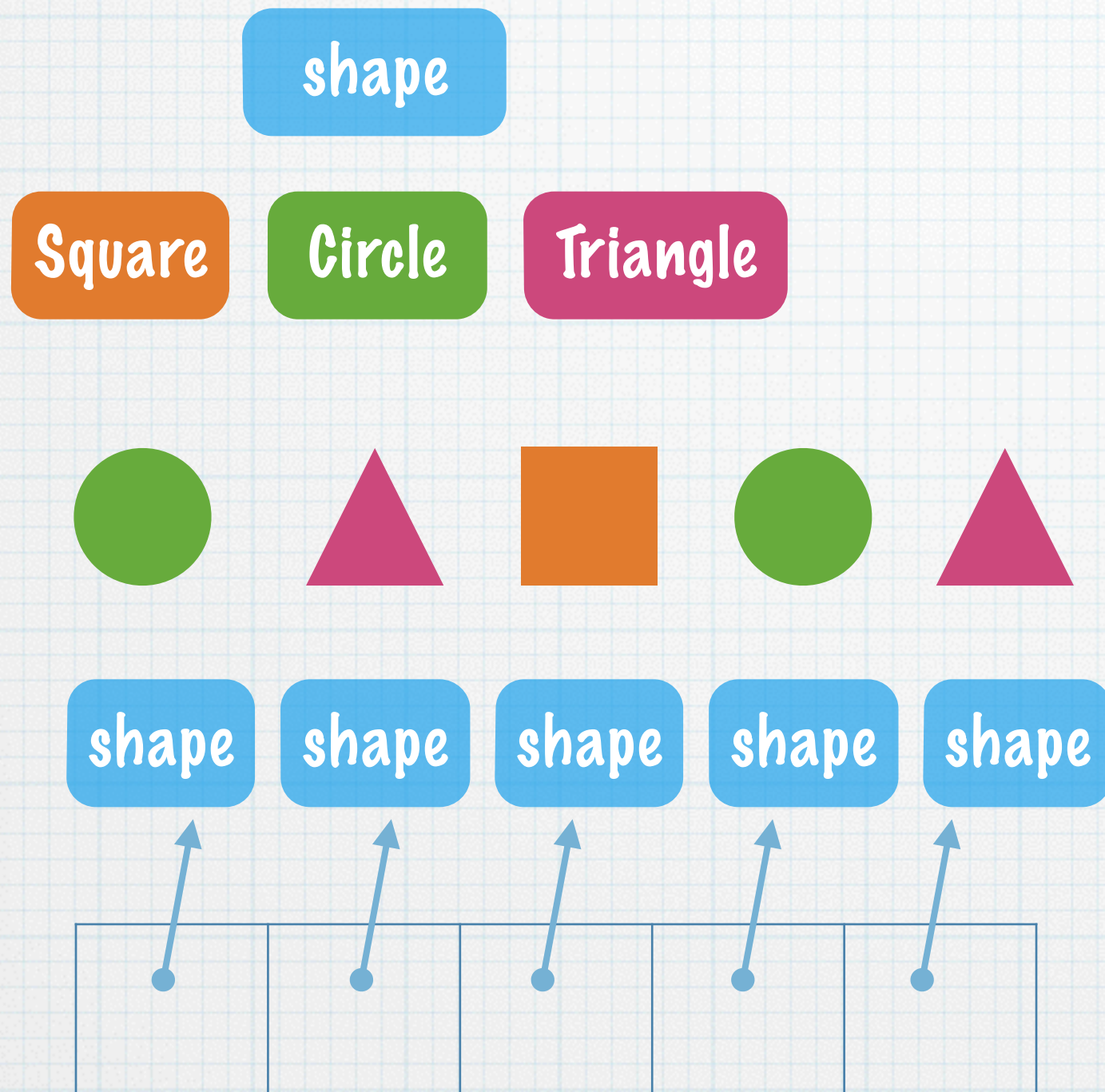


```
for ( auto s : shapes )  
    s->draw();
```

`Vector<shape*> shapes`



# Polymorfi



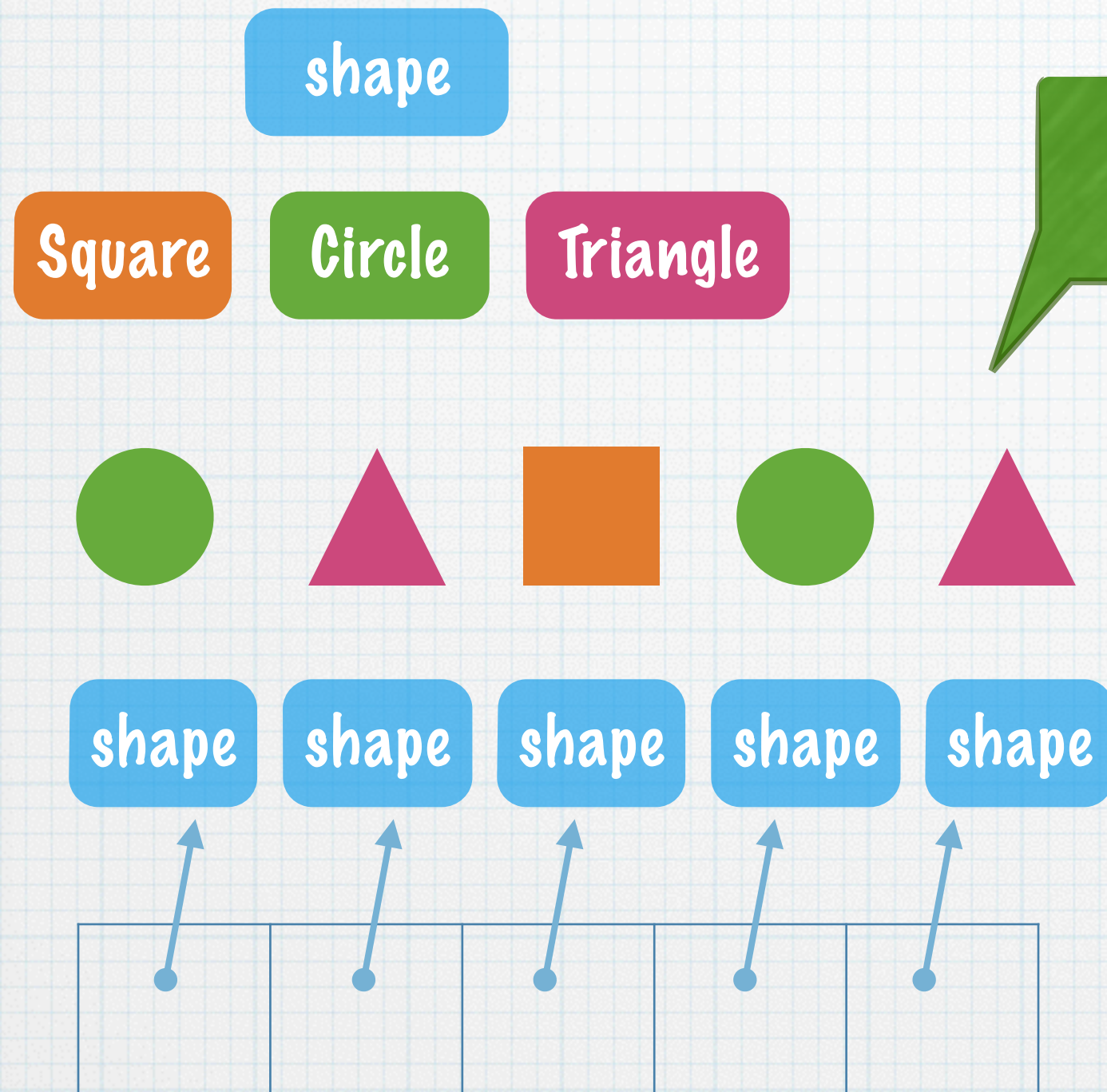
Samme type pekere

```
for ( auto s : shapes )  
    s->draw();
```

`Vector<shape*> shapes`



# Polymorfi



Mange former -  
«Poly» «Morf»

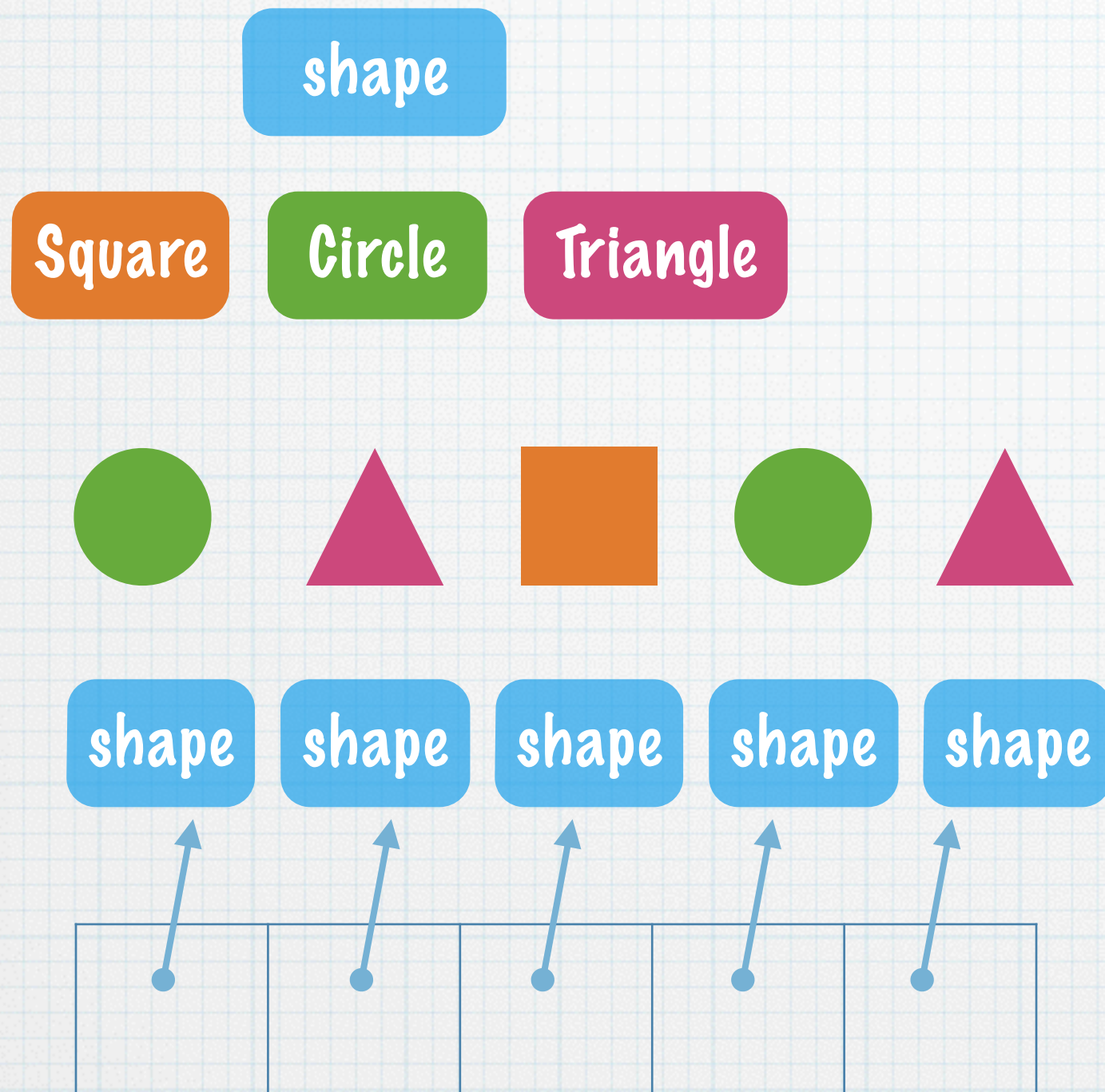
Samme type pekere

```
for ( auto s : shapes )  
    s->draw();
```

```
Vector<shape*> shapes
```



# Polymorfi



`Vector<shape*> shapes`

En «polymorf peker» kan lages slik:

```
shape* s1 = new Triangle;
```

Eller lokalt (men OBS):

```
Triangle t;  
shape* s2 = &t
```

```
for ( auto s : shapes )  
    s->draw();
```



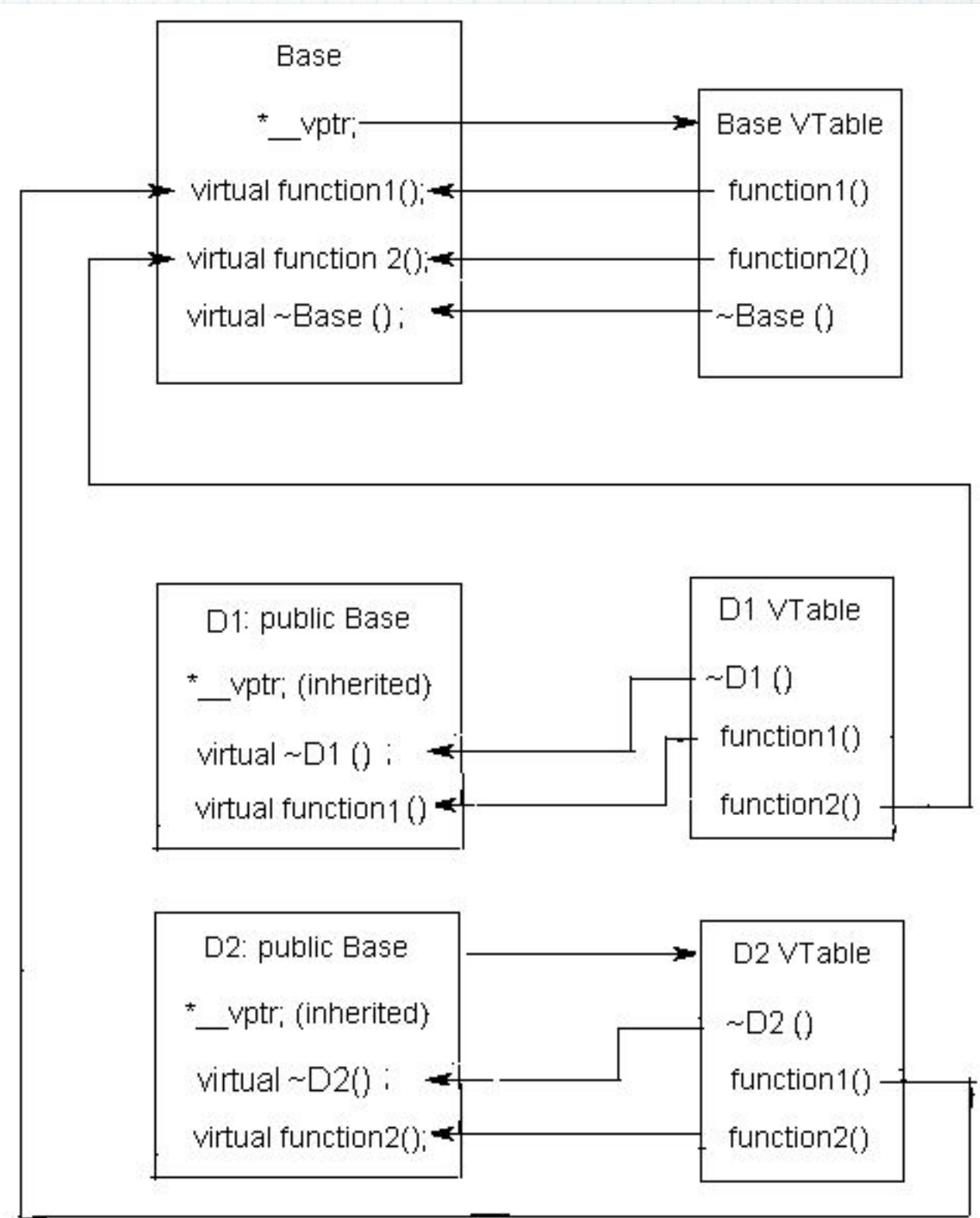
# Polymorfi

- \* Man kan bruke nøkkelordet **virtual** også i subklasser, men ikke nødvendig. **override** anbefales i stedet (C++ 11) - da kan kompilatoren hjelpe deg.
- \* Polymorfi funker for pekere, og referanser
- \* Ulempen med å bruke kun en referanse er at man gjerne har brukt new for å lage pekeren - mindre tydelig at man må "delete" en referanse
- \* Hvis man ikke bruker **virtual** kan **B** og **S** fint ha funksjonen **f** men den er da ikke "overridet" og funker kun når man har en peker av subtype.
- \* Klasser «der polymorfi er naturlig» bør ha virtuelle destruktorer... Hvorfor?
  - \* Anta `shape* s = new triangle;` der `triangle` har «new data»
  - \* så «`delete s....`». Nå kalles destructor i `s` - hva med destructor i `triangle`?
- \* **OBS:** Virtuelle funksjoner har litt overhead (vtbl / vtable).
  - \* Men kompilatoren klarer \*ofte\* å unngå den helt.



# vtbl

- \* For hver klasse med virtuelle funksjoner finnes en «virtuell tabell», vtbl
- \* Her er det pekere til de funksjonene som gjelder for klassen
- \* Hver klasse har et ekstra skjult medlem, vptr, som peker til riktig vtbl
- \* Hvert funksjonskall medfører et oppslag
  - \* ca. en dereferens
- \* se Boka s.506





# vtbl

- \* For hver klasse med virtuelle funksjoner finnes en «virtuell tabell», vtbl

- \* Her er det pekere til de funksjonene som gjelder for klassen

- \* Hver klasse har et ekstra skjult medlem, vptr, som peker til riktig vtbl

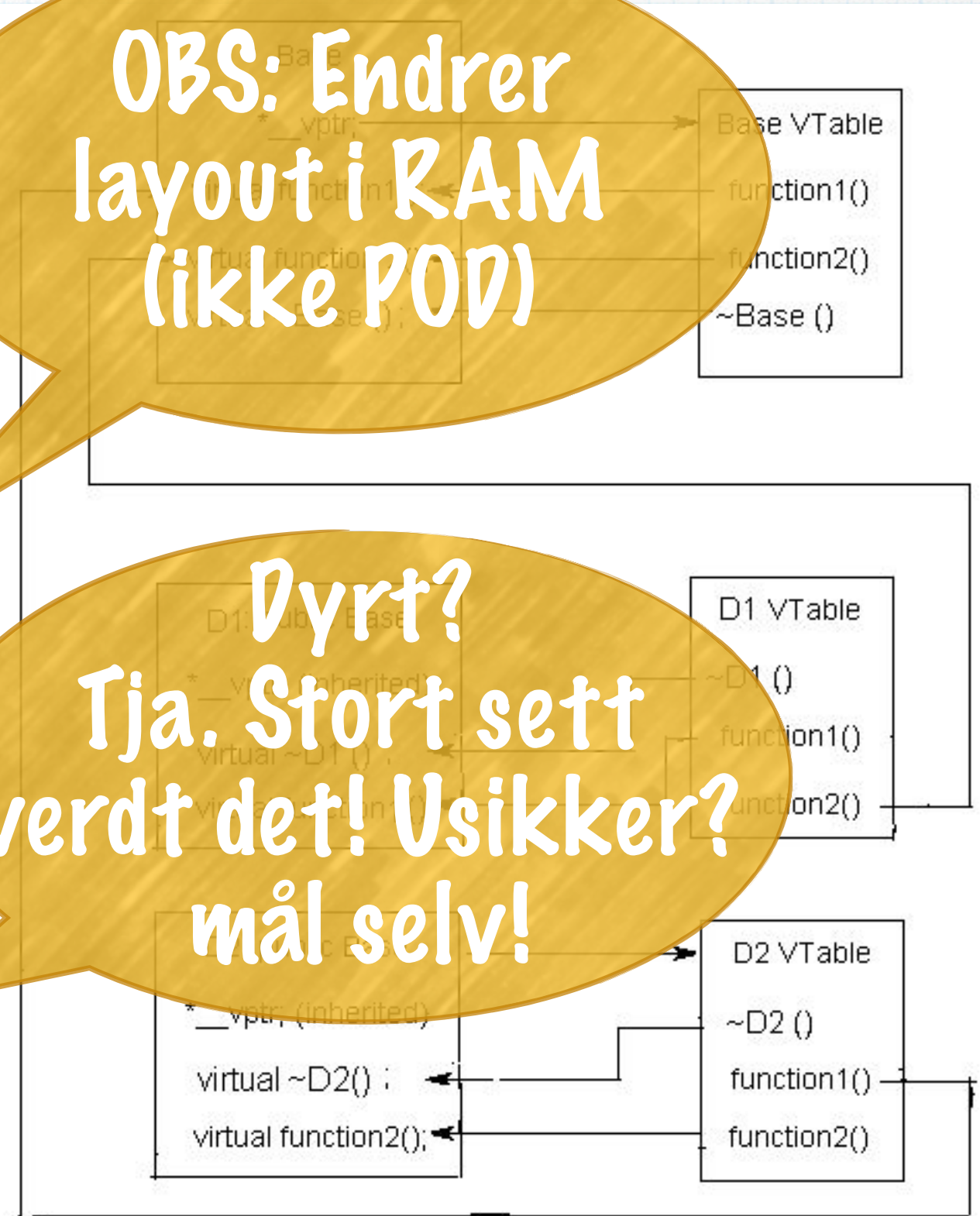
- \* Hvert funksjonskall medfører et oppslag

- \* ca. en dereferens

- \* se Boka s.506

OBS: Endrer layout i RAM (ikke POD)

Dyrt?  
Tja. Stort sett verdt det! Usikker? må selv!





# Demo:

---

`polymorfi1.cpp`

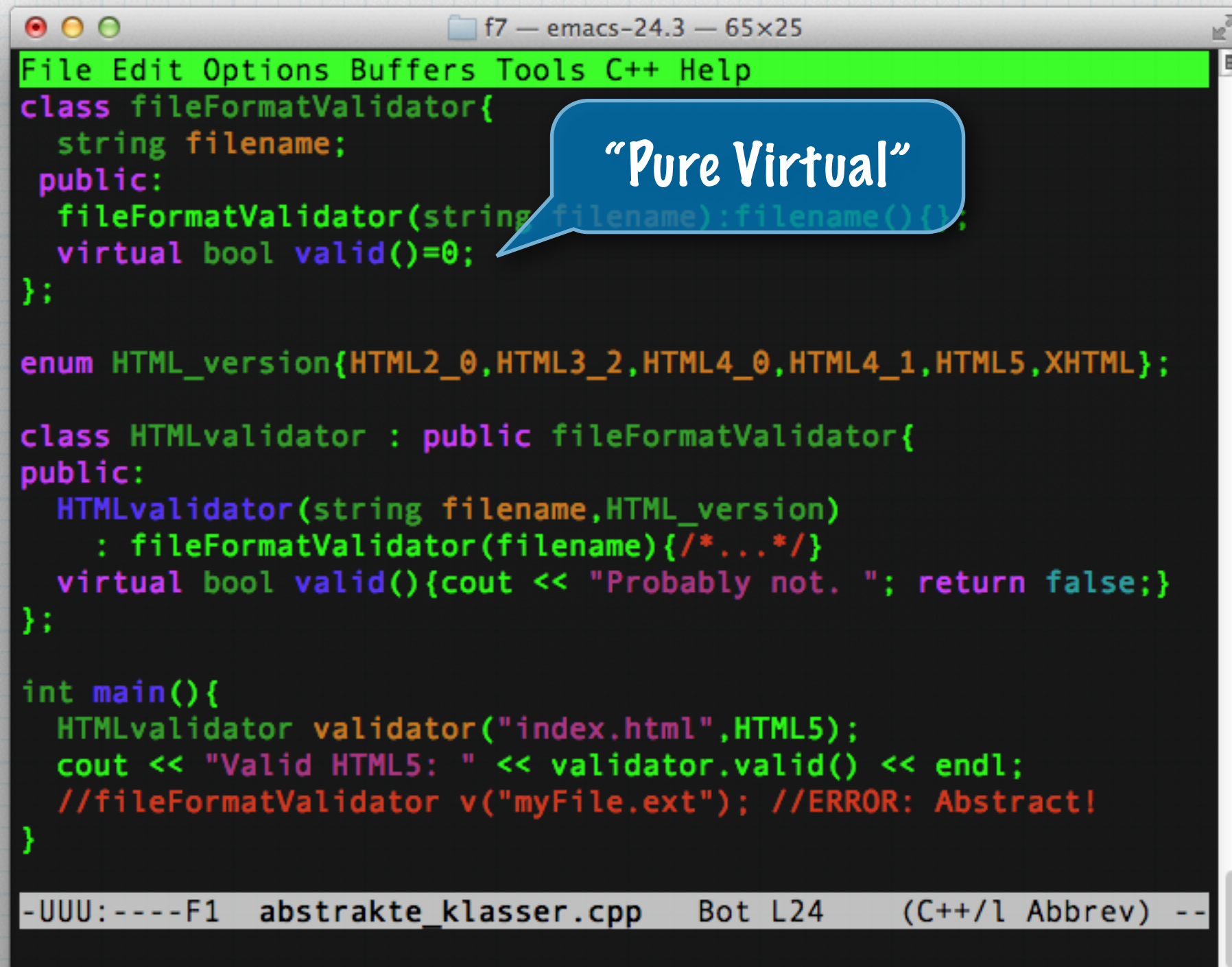


# Abstrakte klasser

- \* En abstrakt klasse er en klasse som ikke kan instansieres direkte (kun via subtype)
- \* I C++ blir en klasse abstrakt hvis den har minst en **virtual** funksjon som ikke kan implementeres i baseklassen
- \* Man definerer en funksjon "pure virtual" ved å «nekte den en kropp», slik:  
**virtual void ask(string question) = 0;**
- \* Abstrakte klasser er nyttige når det å snakke om instanser av klassen blir "for generelt" til å gi mening. Feks:
  - \* "fileFormatCheker" (hvilket format?)
  - \* "connection" (hva slags?)



# Abstrakte klasser



```
f7 — emacs-24.3 — 65x25
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){};
    virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

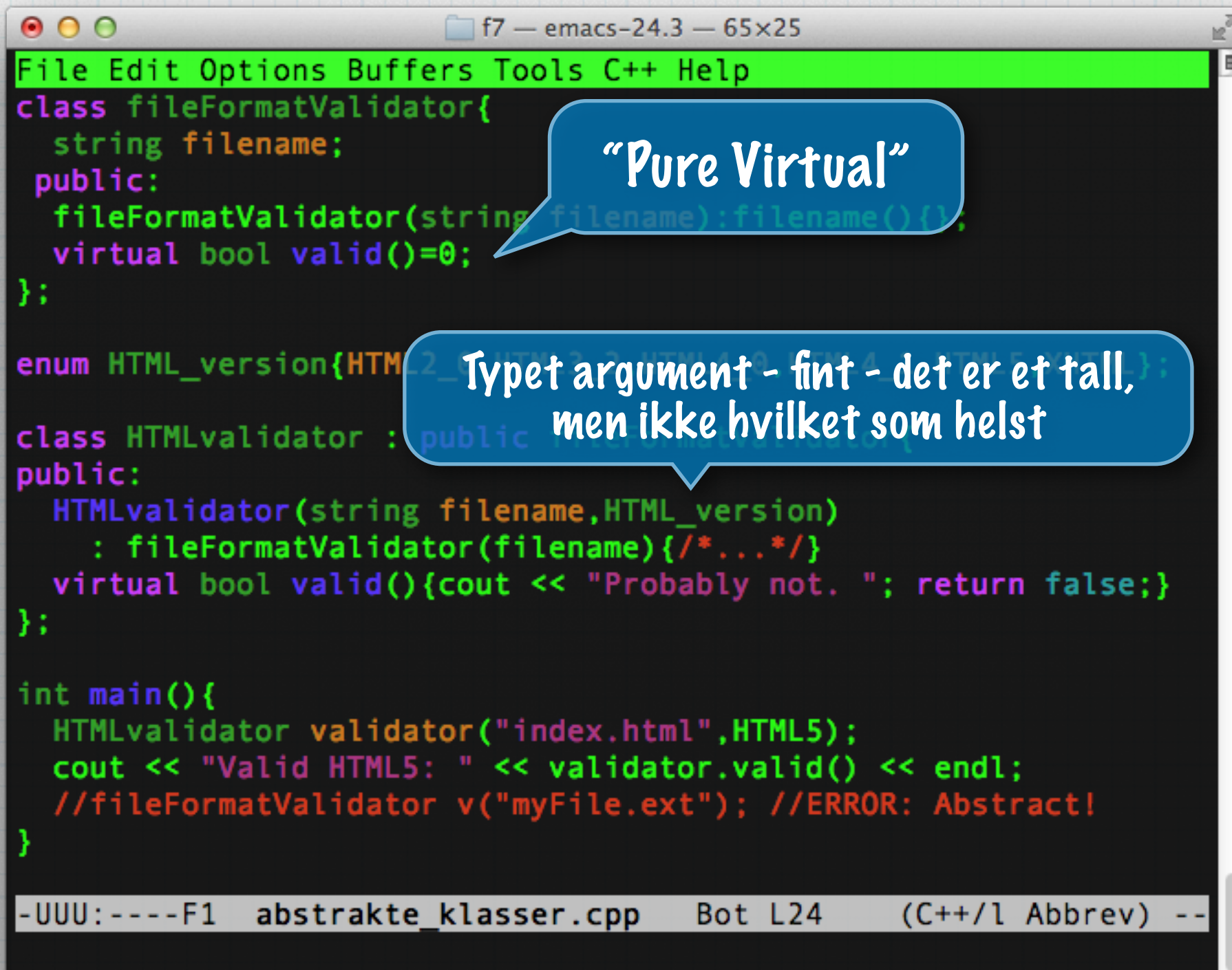
class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html",HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}

-UUU:----F1  abstrakte_klasser.cpp  Bot L24  (C++/l Abbrev) --
```



# Abstrakte klasser



```
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){};
    virtual bool valid()=0;
};

enum HTML_version{HTML_2_0, HTML_3_2, HTML_4_0, HTML_4_01, HTML_5};

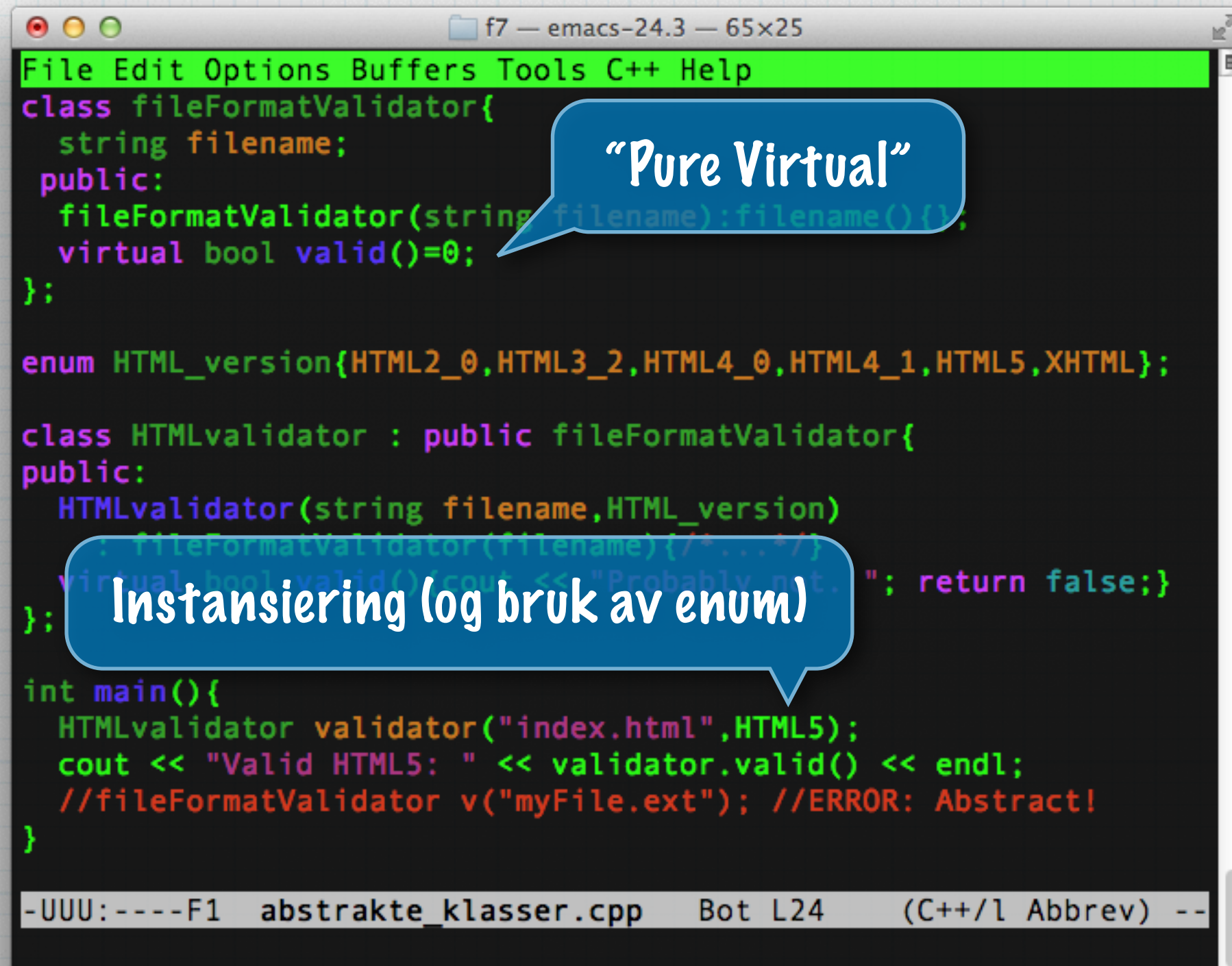
class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename, HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html", HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}

-UUU:----F1 abstrakte_klasser.cpp Bot L24 (C++/l Abbrev) --
```



# Abstrakte klasser



```
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){};
    virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html",HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}

-UUU:----F1 abstrakte_klasser.cpp Bot L24 (C++/l Abbrev) --
```



# Abstrakte klasser

```
f7 — emacs-24.3 — 65x25
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){};
    virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html",HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}

-UUU:----F1  abstrakte_klasser.cpp  Bot L24  (C++/l Abbrev) --
```



Abstrakt?

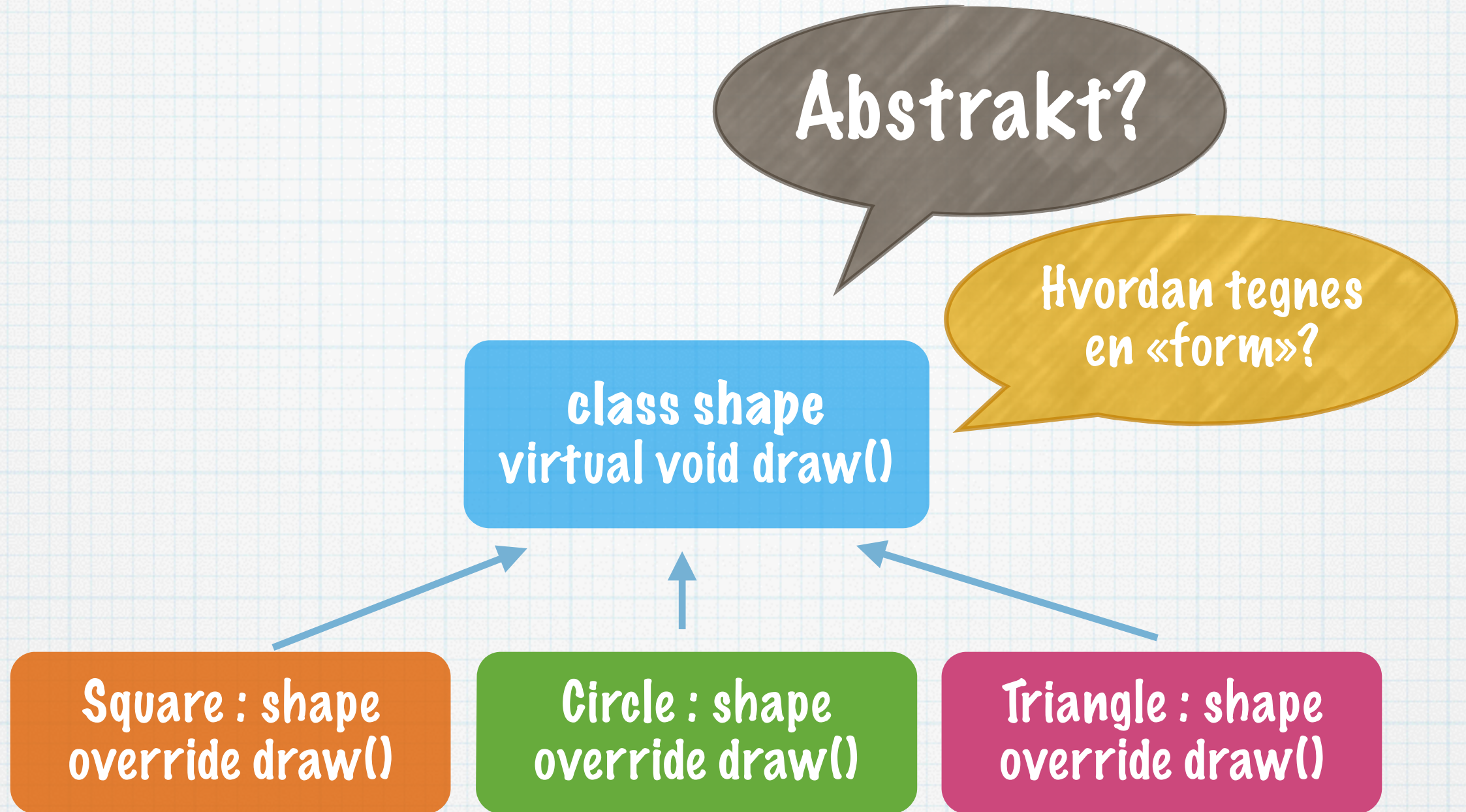
Hvordan tegnes  
en «form»?

```
class shape  
virtual void draw()
```

```
Square : shape  
override draw()
```

```
Circle : shape  
override draw()
```

```
Triangle : shape  
override draw()
```





# Demo:

---

`abstrakte_klasser.cpp`

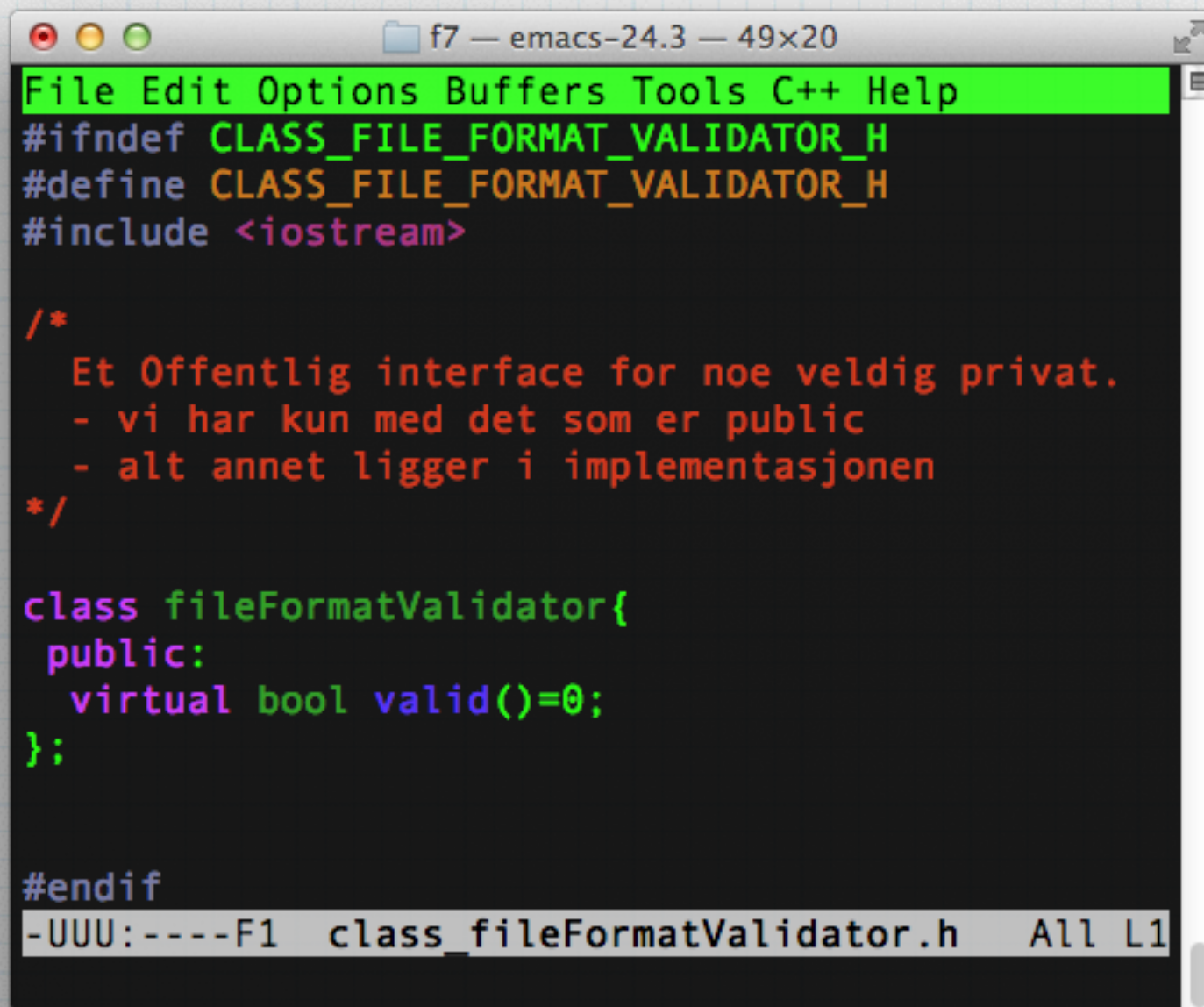


# Abstrakte klasser som interface

- \* Abstrakte klasser kan brukes som "interface" i Java:
  - \* Effektivt sett er de en "liste" av ting som må være med
  - \* En klasse kan arve mange (multippel arv)
  - \* Trenger abstrakte klasser noe privat?
    - \* Ikke nødvendigvis - men noen medlemmer er ofte nødvendig (feks. "filename", "username" etc.)
    - \* Og: man kan gjerne implementere funksjoner i en abstrakt klasse. De kan bare ikke brukes før du har en subklasse. Nyttig?
    - \* Ja: feks. funksjoner som "toString" kan gjerne ha en "default"
  - \* Hvis man er "microsoft" og kun ønsker å vise frem det som er public?
  - \* Da kan man lage en abstrakt baseklasse over den første, som bare har med det som er public. Resten kan puttes i implementasjonen.



# Abstrakte klasser som interface



```
f7 — emacs-24.3 — 49x20
File Edit Options Buffers Tools C++ Help
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
#include <iostream>

/*
  Et Offentlig interface for noe veldig privat.
  - vi har kun med det som er public
  - alt annet ligger i implementasjonen
*/

class fileFormatValidator{
public:
  virtual bool valid()=0;
};

#endif
-UUU:----F1  class_fileFormatValidator.h  All L1
```

Hvis man er "Microsoft" vil man kanskje eksponere minst mulig av koden.

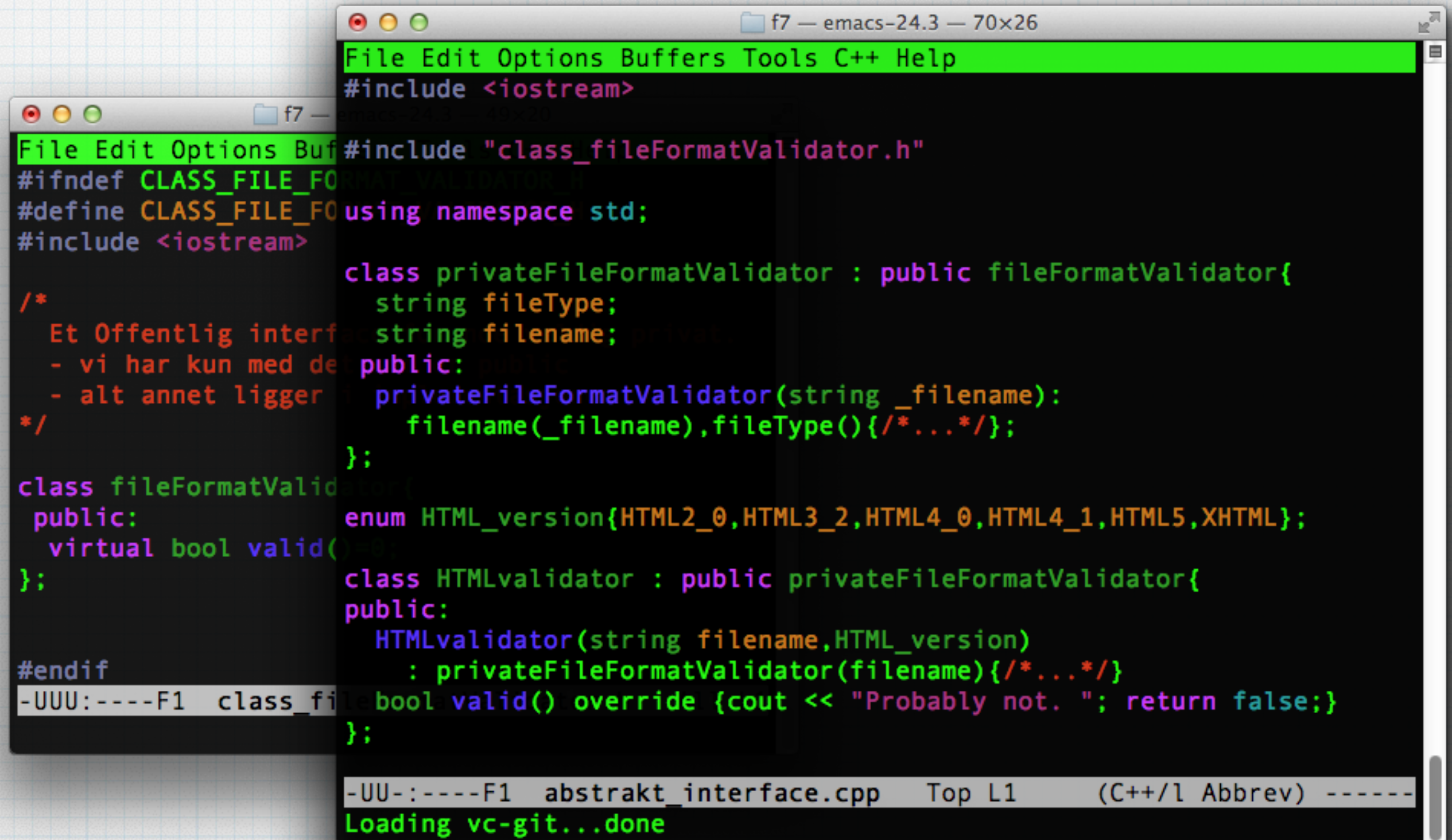
Men: Vi vil at folk skal kunne bruke bibliotekene våre, så vi må ha en headerfil

Løsning:  
Enda et nivå av arv

Enda et nivå av arv  
Løsning:



# Abstrakte klasser som interface



The image shows two Emacs windows. The background window displays the header file `class_fileFormatValidator.h`, which defines an abstract class `fileFormatValidator` and a private implementation class `privateFileFormatValidator`. The foreground window displays the implementation file `abstrakt_interface.cpp`, which includes the header and implements the `valid()` method for `HTMLvalidator`.

```
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
using namespace std;
#include <iostream>

/*
  Et Offentlig interface
  - vi har kun med de virtuelle metoder
  - alt annet ligger i privateFileFormatValidator
*/

class fileFormatValidator{
public:
  virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public privateFileFormatValidator{
public:
  HTMLvalidator(string filename,HTML_version
    : privateFileFormatValidator(filename){/*...*/}
  bool valid() override {cout << "Probably not. "; return false;}
};

#endif
-UUU:----F1 class_fileFormatValidator.h
```

```
f7 — emacs-24.3 — 70x26
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
using namespace std;

class privateFileFormatValidator : public fileFormatValidator{
  string fileType;
  string filename;
public:
  privateFileFormatValidator(string _filename):
    filename(_filename),fileType(){/*...*/};
};

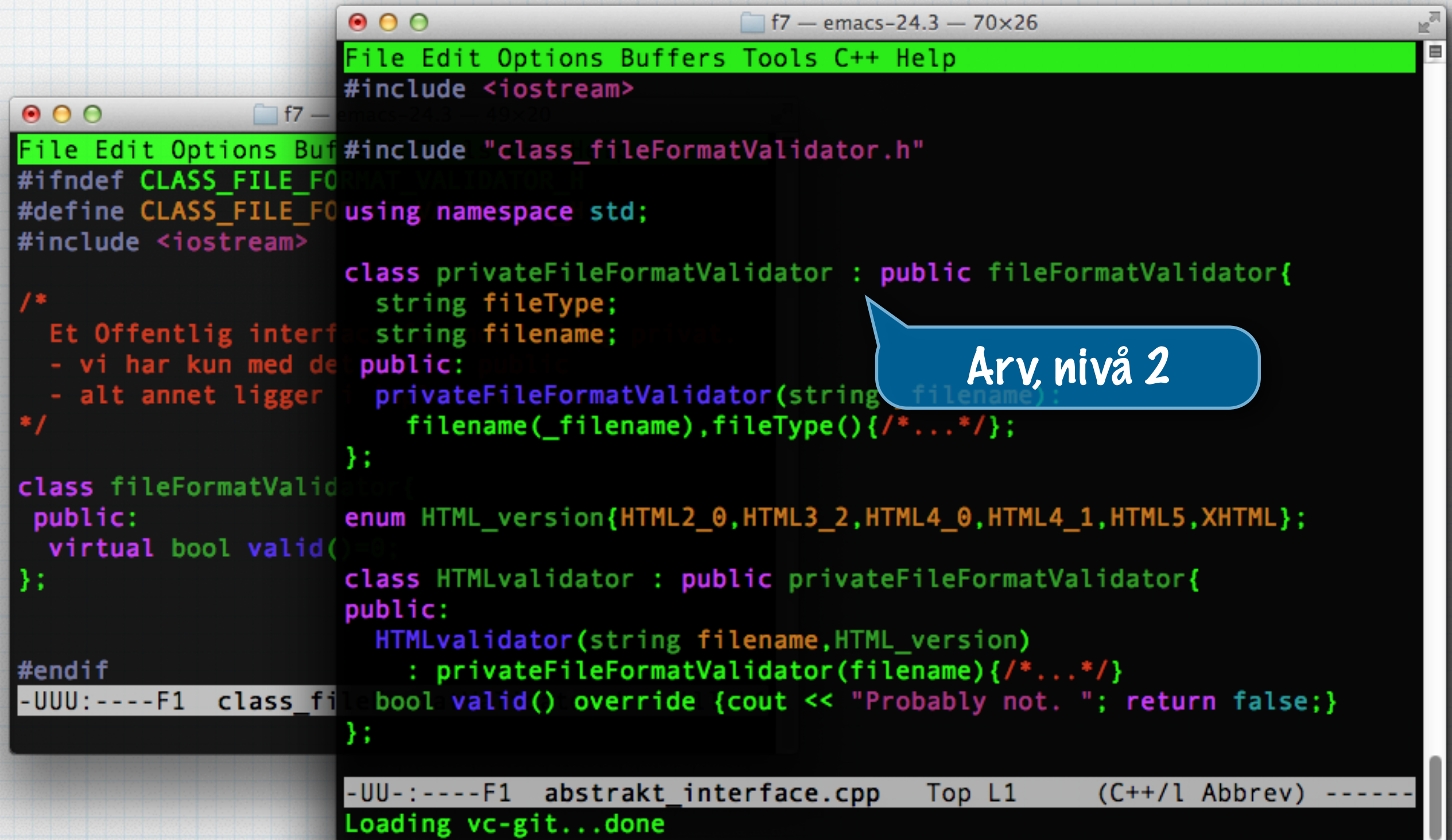
enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public privateFileFormatValidator{
public:
  HTMLvalidator(string filename,HTML_version
    : privateFileFormatValidator(filename){/*...*/}
  bool valid() override {cout << "Probably not. "; return false;}
};

-UU-:----F1 abstrakt_interface.cpp Top L1 (C++/1 Abbrev) -----
Loading vc-git...done
```



# Abstrakte klasser som interface



The image shows two Emacs windows. The background window displays the header file `class_fileFormatValidator.h`, which defines an abstract class `fileFormatValidator` and its inheritance hierarchy. The foreground window displays the implementation file `abstrakt_interface.cpp`, showing a concrete implementation of the `fileFormatValidator` interface.

```
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
using namespace std;

class privateFileFormatValidator : public fileFormatValidator{
    string fileType;
    string filename;
public:
    privateFileFormatValidator(string filename):
        filename(_filename),fileType(){/*...*/};
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public privateFileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : privateFileFormatValidator(filename){/*...*/}
    bool valid() override {cout << "Probably not. "; return false;}
};

#endif
-UUU:----F1 class_fileFormatValidator.h
```

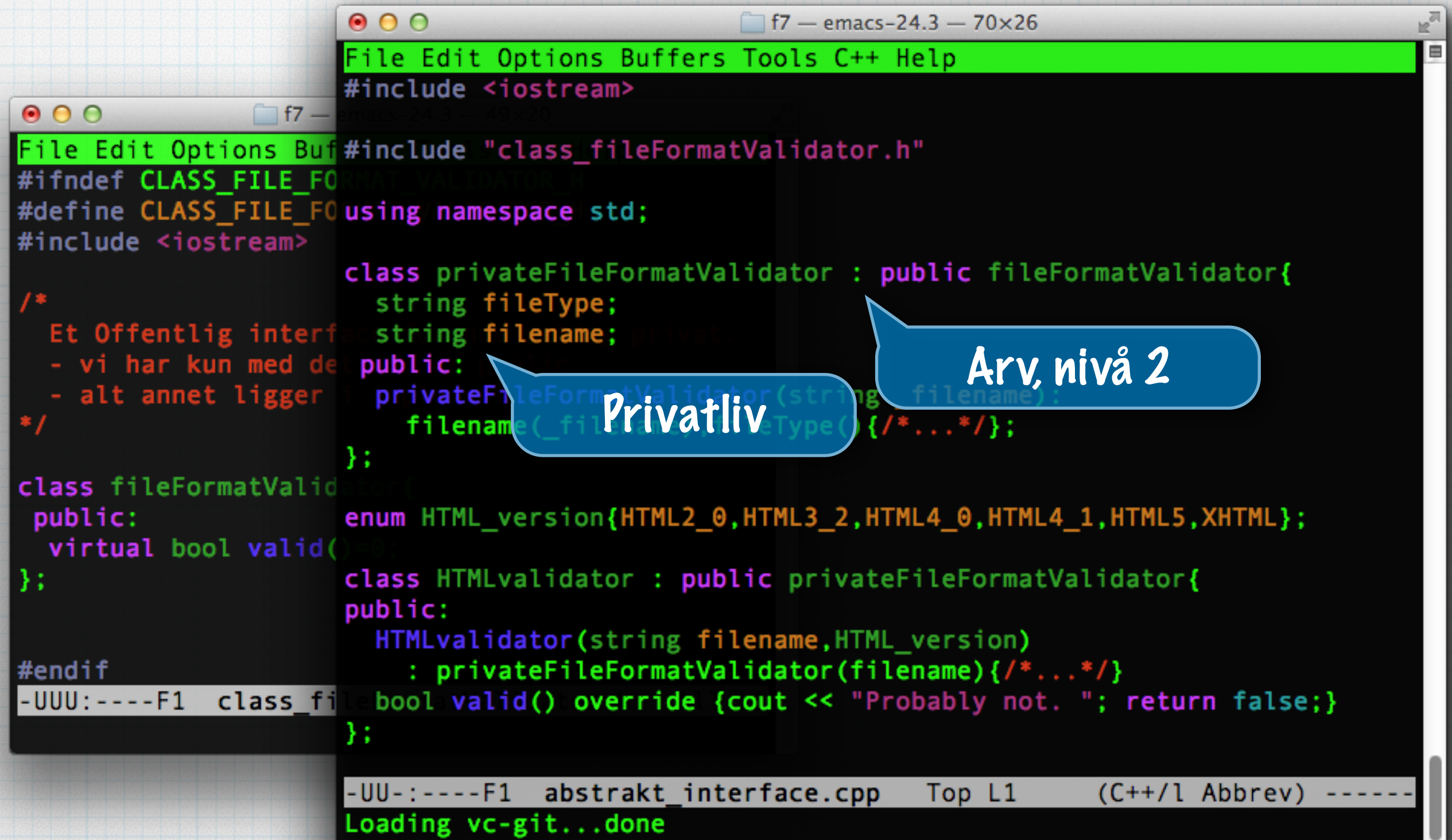
Et Offentlig interface  
- vi har kun med de  
- alt annet ligger

Arv, nivå 2

```
-UU-:----F1 abstrakt_interface.cpp Top L1 (C++/1 Abbrev) -----
Loading vc-git...done
```



# Abstrakte klasser som interface



The image shows two Emacs windows displaying C++ code. The left window shows the header file `class_fileFormatValidator.h`, and the right window shows the implementation file `abstrakt_interface.cpp`. The code defines an abstract class `fileFormatValidator` and its inheritance hierarchy: `privateFileFormatValidator` (private) and `HTMLvalidator` (public).

```
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
using namespace std;
#include <iostream>

/*
Et Offentlig interface
- vi har kun med de public:
- alt annet ligger i privatliv
*/

class fileFormatValidator{
public:
virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public privateFileFormatValidator{
public:
HTMLvalidator(string filename,HTML_version
: privateFileFormatValidator(filename){/*...*/}
bool valid() override {cout << "Probably not. "; return false;}
};

#endif
-UUU:----F1 class_fileFormatValidator.h
```

Arv, nivå 2

Privatliv

```
f7 — emacs-24.3 — 70x26
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
using namespace std;
class privateFileFormatValidator : public fileFormatValidator{
string fileType;
string filename;
privateFileFormatValidator(string filename):
filename(_filename),fileType(){/*...*/};
};

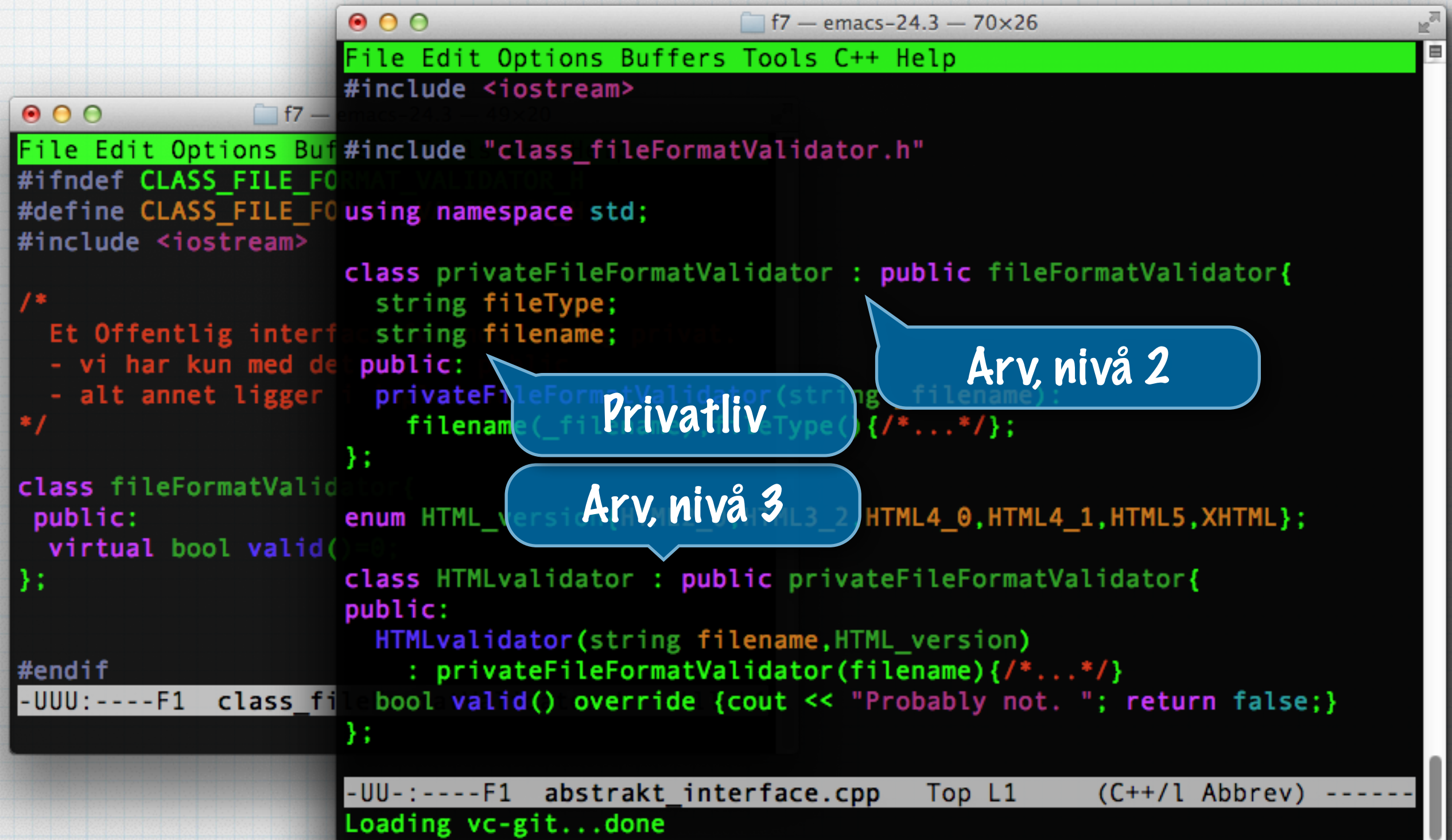
enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public privateFileFormatValidator{
public:
HTMLvalidator(string filename,HTML_version
: privateFileFormatValidator(filename){/*...*/}
bool valid() override {cout << "Probably not. "; return false;}
};

-UU-:----F1 abstrakt_interface.cpp Top L1 (C++/1 Abbrev) -----
Loading vc-git...done
```



# Abstrakte klasser som interface



The image shows two Emacs windows displaying C++ code. The left window shows the header file `class_fileFormatValidator.h` with an abstract class `fileFormatValidator`. The right window shows the implementation file `abstrakt_interface.cpp` with two derived classes: `privateFileFormatValidator` and `HTMLvalidator`. Annotations highlight the inheritance levels: 'Privatliv' for the private class, 'Arv, nivå 2' for the first derived class, and 'Arv, nivå 3' for the second derived class.

```
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
using namespace std;
#include <iostream>

/*
Et Offentlig interface
- vi har kun med de som er offentlig
- alt annet ligger i private
*/

class fileFormatValidator {
public:
    virtual bool valid() = 0;
};

#endif

-UUU:----F1 class_fileFormatValidator.h

File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
using namespace std;

class privateFileFormatValidator : public fileFormatValidator {
    string fileType;
    string filename;
public:
    privateFileFormatValidator(string filename):
        filename(filename), fileType("HTML3_2") { /*...*/ };
};

enum HTML_version {HTML3_2, HTML4_0, HTML4_1, HTML5, XHTML};

class HTMLvalidator : public privateFileFormatValidator {
public:
    HTMLvalidator(string filename, HTML_version version)
        : privateFileFormatValidator(filename) { /*...*/ }
    bool valid() override { cout << "Probably not. "; return false; }
};

-UU-:----F1 abstrakt_interface.cpp Top L1 (C++/1 Abbrev) -----
Loading vc-git...done
```



# Abstrakte klasser som interface

The image shows a screenshot of the Emacs editor with two windows displaying C++ code. The background window shows the definition of an abstract class `fileFormatValidator`. The foreground window shows two subclasses: `privateFileFormatValidator` and `HTMLvalidator`.

**Annotations:**

- Privatliv**: Points to the `private` access specifier in the `privateFileFormatValidator` class definition.
- Arv, nivå 2**: Points to the `public fileFormatValidator` inheritance in the `privateFileFormatValidator` class definition.
- Arv, nivå 3**: Points to the `HTMLvalidator` class definition, which inherits from `privateFileFormatValidator`.
- ordet «override» hindrer feilstaving av funksjonsnavn**: Points to the `override` keyword in the `bool valid()` method of `HTMLvalidator`.

**Code Snippets:**

**Background Window (abstract\_fileFormatValidator.h):**

```
#include <iostream>

#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
using namespace std;

class fileFormatValidator {
public:
    virtual bool valid() = 0;
};

#endif
```

**Foreground Window (abstract\_interface.cpp):**

```
#include "class_fileFormatValidator.h"

class privateFileFormatValidator : public fileFormatValidator {
    string fileType;
    string filename;
public:
    privateFileFormatValidator(string filename):
        filename(filename), fileType("HTML") { /*...*/ };
};

enum HTML_version {HTML3_2, HTML4_0, HTML4_1, HTML5, XHTML};

class HTMLvalidator : public privateFileFormatValidator {
public:
    HTMLvalidator(string filename) : privateFileFormatValidator(filename) { /*...*/ };
    bool valid() override { cout << "Probably not. "; return false; };
};
```

**Status Bar:**

-UU-:----F1 abstrakt\_interface.cpp Top L1 (C++/1 Abbrev) -----  
Loading vc-git...done



# Abstrakte klasser som interface

**OBS:**

I kurset skal alle klasser defineres i egne headerfiler. Her har vi gjort det enkelt, for eksemplets skyld.

**OBS OBS:**

Vi oppmuntret ikke til denne doble arven i kurset, men det er nødvendig/nyttig for noen.

```
f7 — emacs-24.3 — 70x26
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
using namespace std;

class privateFileFormatValidator : public fileFormatValidator{
    string fileType;
    string filename;
public:
    privateFileFormatValidator(string filename):
        filename(_filename), fileType("/*...*/");
};

enum HTML_version{HTML3_2, HTML4_0, HTML4_1, HTML5, XHTML};

class HTMLvalidator : public privateFileFormatValidator{
public:
    HTMLvalidator(string filename, HTML_version
        : privateFileFormatValidator(filename){/*...*/}
    bool valid() override {cout << "Probably not. "; return false;}
};

-UU-:----F1 abstrakt_interface.cpp Top L1 (C++/1 Abbrev) -----
Loading vc-git...done
```

Privatliv

Arv, nivå 2

Arv, nivå 3



# Demo:

---

`abstrakt_interface.cpp`  
`class_fileFormatValidator.h`