

Effektiv Kode med C og C++

Forelesning 5, vår 2014
Alfred Bratterud

Agenda:

- * Litt repetisjon
 - * Constructorer og destructorer
- * Klasser og headerfiler
- * Makefile
- * Exceptions
- * Iteratorer
- * Oppklaring: Map og nøkler

Litt repetisjon

konstruktorer i klasser

- * Konstruktoren har som jobb å initialisere alle medlemmer - ved å kalle deres konstruktorer.
- * OBS: Lager du en konstruktur selv, uten argumenter, mister du standard konstruktoren.
- * I alle egne konstruktorer må alle medlemmer initialiseres manuelt - evt. ved å kalle standard-konstruktur.
- * Man må da initialisere slik:

```
class Student{  
    int nr;  
    string name;  
    public:  
        student(string name): nr(10), name(name) { ... }  
        int get_nr();  
        string get_name();  
};
```

Destruktorer

- * Destruktor har ansvar for å rydde opp, dvs. frigjøre det minnet klassen har satt av.
- * Destruktoren til en klasse “**myClass**” heter “
~myClass()
- * Alle objekter har en “default destructor”
- *og den fjerner sizeof(object)
- * Destruktorer er viktige når vi har brukt **new** i klassen
- * Språk med garbage-collection trenger ikke destruktorer - men de bruker da tid på søppeltømming

Interface v.s. Implementation

- * “Interface” til en klasse (eller et bibliotek) består av deklarasjoner av alle medlemmene, men kun med “signaturene” til funksjonene
 - * Denne ligger gjerne i en egen “header-fil” (Student.h)
 - * I C kalles header-fila gjerne “interface”. Tenk API.
 - * I C++ refererer man strengt tatt til det som er public, som et interface. Men er det mulig å kun vise det som er public i en headerfil?
- * “Implementasjonen” ligger gjerne i en annen fil (Student.cpp), som “inkluderer” header-fila
- * Fordel: da kan vi kompile - og bruke bit for bit separat.
- * ...og skifte ut ferdig-kompileerde “biter” uten å rekompile alt.

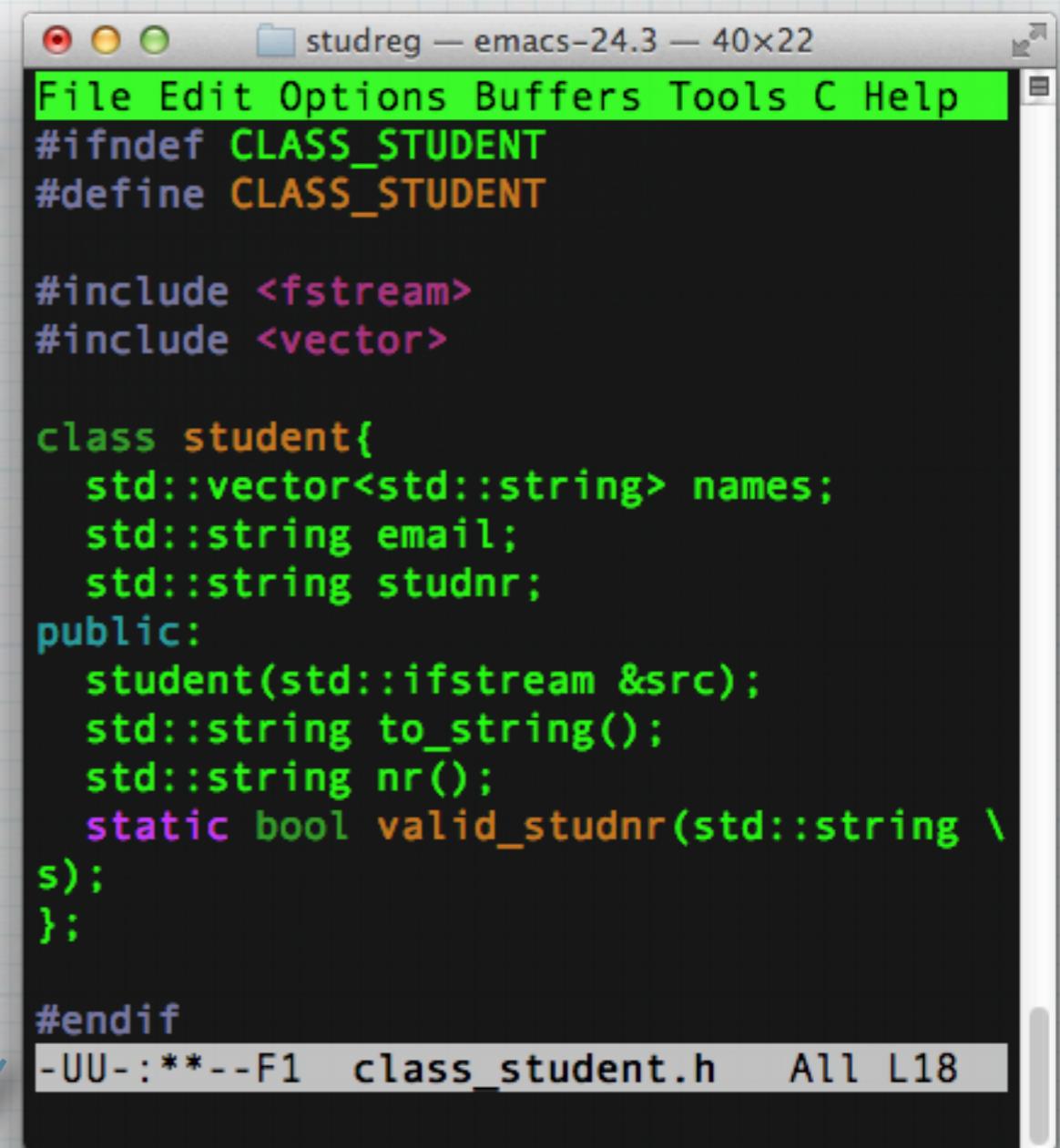
Klassestandard i kurset

- * Dette gjelder fom. oblig2:
- * Klasser skal ha to filer, en ".h" og en ".cpp"
- * For en klasse, "myclass" skal filene hete "myclass.h" og "myclass.cpp"
- * "myclass.h" skal ikke inneholde noe kjørbar kode. Konstanter er OK, men da i eget namespace.
- * Vi lager "pakker" ved å definere namespaces - du kan gjerne ha flere

Kursstandard: header

“Include guard”:
En macro som hindrer
multippel inkludering

“Include guard”



The screenshot shows a window titled "studreg — emacs-24.3 — 40x22". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "C", and "Help". The code in the buffer is:

```
#ifndef CLASS_STUDENT
#define CLASS_STUDENT

#include <fstream>
#include <vector>

class student{
    std::vector<std::string> names;
    std::string email;
    std::string studnr;
public:
    student(std::ifstream &src);
    std::string to_string();
    std::string nr();
    static bool valid_studnr(std::string \
s);
};

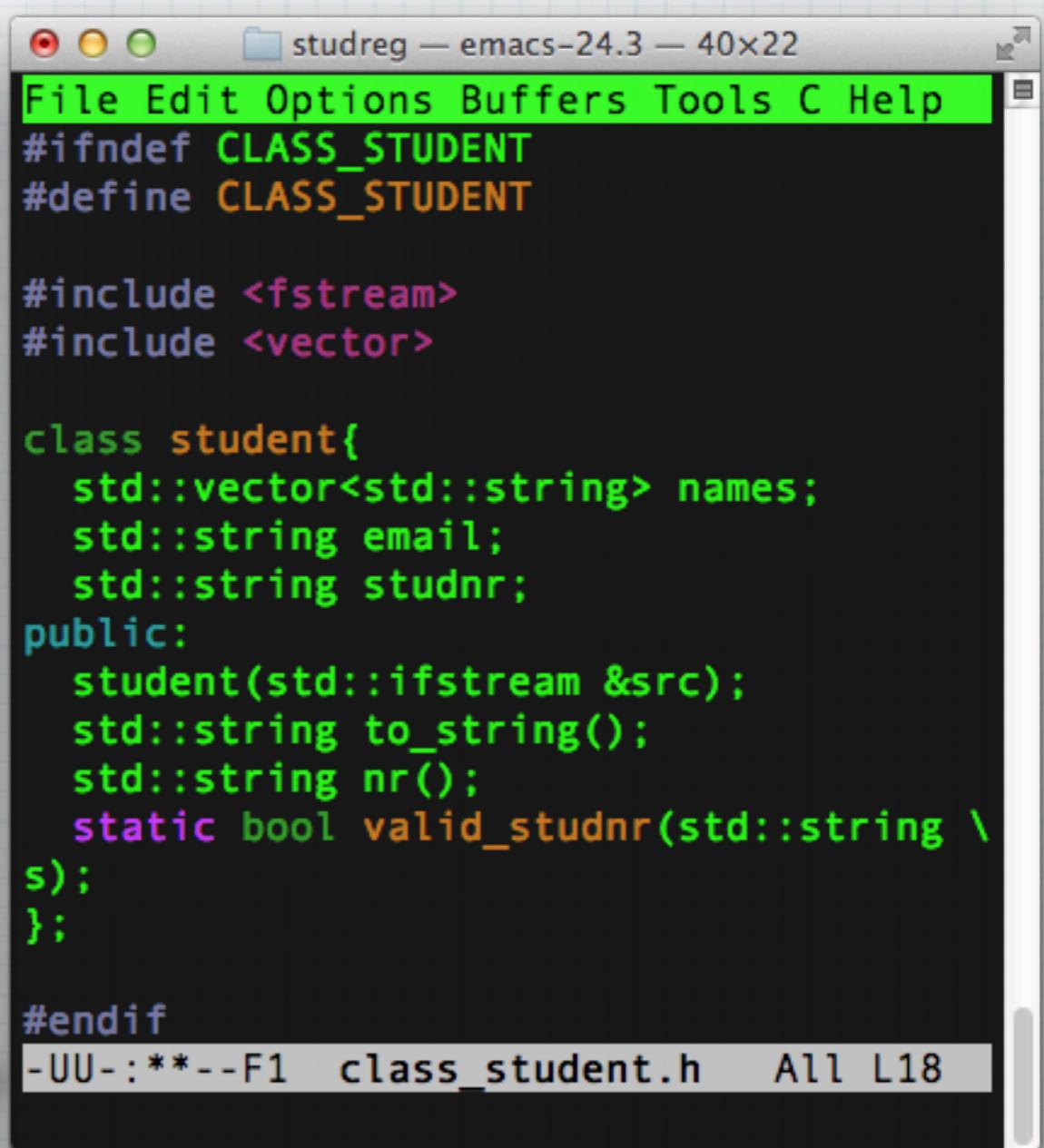
#endif
-UU-:***-F1  class_student.h  All L18
```

Kursstandard: header

“Include guard”:
En macro som hindrer
multippel inkludering

Ingen “using namespace std;”

“Include guard”



The screenshot shows an Emacs window titled "studreg — emacs-24.3 — 40x22". The menu bar includes File, Edit, Options, Buffers, Tools, C, and Help. The code buffer contains the following C++ code:

```
#ifndef CLASS_STUDENT
#define CLASS_STUDENT

#include <iostream>
#include <vector>

class student{
    std::vector<std::string> names;
    std::string email;
    std::string studnr;
public:
    student(std::ifstream &src);
    std::string to_string();
    std::string nr();
    static bool valid_studnr(std::string \
s);
};

#endif
-UU-:***-F1  class_student.h  All L18
```

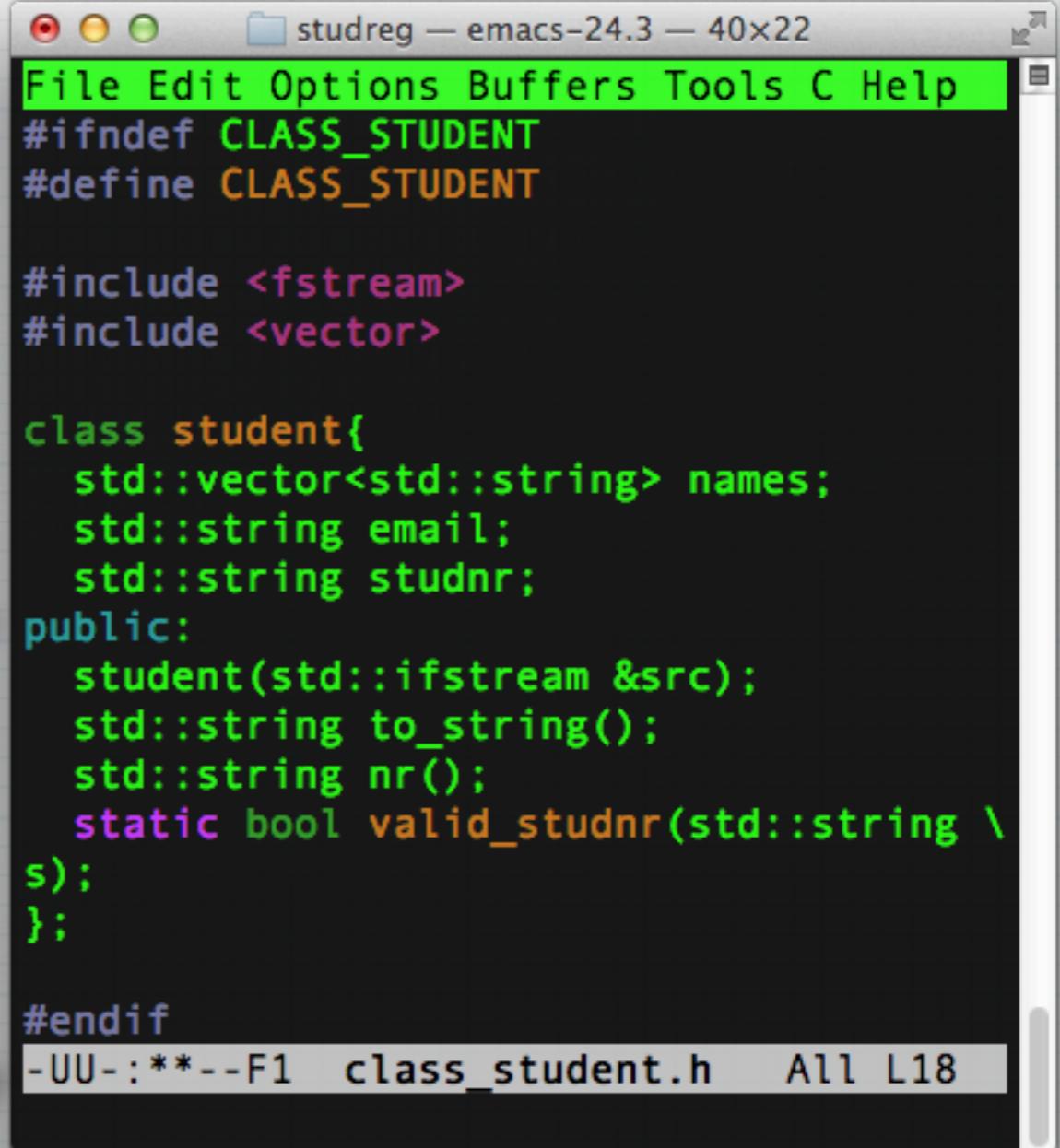
Kursstandard: header

“Include guard”:
En macro som hindrer
multippel inkludering

Ingen “using namespace std;”

En klasse, helt uten kropper.
Men- vi har med includes
for å kunne bruke typene
som medlemmer

“Include guard”



The screenshot shows an Emacs window titled "studreg — emacs-24.3 — 40x22". The menu bar includes File, Edit, Options, Buffers, Tools, C, and Help. The code in the buffer is:

```
#ifndef CLASS_STUDENT
#define CLASS_STUDENT

#include <iostream>
#include <vector>

class student{
    std::vector<std::string> names;
    std::string email;
    std::string studnr;
public:
    student(std::ifstream &src);
    std::string to_string();
    std::string nr();
    static bool valid_studnr(std::string \
s);
};

#endif
```

The status bar at the bottom right shows "-UU- :***-F1 class_student.h All L18".

Kursstandard: implementasjon

```
studreg — emacs-24.3 — 40x22
File Edit Options Buffers Tools C++ Help
#include "class_student.h"

#include <iostream>
#include <stdexcept>

using namespace std;

string student::nr(){return studnr;}

string student::to_string(){
    return names.front()+" "
        +names.back()+" \nMail: "+email;
}

student::student(ifstream &file):
    email(),names(),studnr()
{
    string s;
    //...
-UU-:***-F1  class_student.cpp   Top L19
```

```
studreg — emacs-24.3 — 40x22
File Edit Options Buffers Tools C Help
#ifndef CLASS_STUDENT
#define CLASS_STUDENT

#include <fstream>
#include <vector>

class student{
    std::vector<std::string> names;
    std::string email;
    std::string studnr;
public:
    student(std::ifstream &src);
    std::string to_string();
    std::string nr();
    static bool valid_studnr(std::string \
s);
};

#endif
-UU-:***-F1  class_student.h   All L18
```

Kursstandard: implementasjon



```
studreg — emacs-24.3 — 40x22
File Edit Options Buffers Tools C++ Help
#include "class_student.h"

#include <iostream>
#include <stdexcept>

using namespace std;

string student::nr(){return studnr;}

string student::to_string(){
    return names.front()+" "
        +names.back()+" \nMail: "+email;
}

student::student(ifstream &file):
    email(),names(),studnr()
{
    string s;
    //...
-UU- :***-F1  class_student.cpp   Top L19
```

```
studreg — emacs-24.3 — 40x22
File Edit Options Buffers Tools C++ Help
#define CLASS_STUDENT

#include <fstream>
#include <vector>

class student{
    std::vector<std::string> names;
    std::string email;
    std::string studnr;
public:
    student(std::ifstream &src);
    std::string to_string();
    std::string nr();
    static bool valid_studnr(std::string \
s);
}

#endif
-UU- :***-F1  class_student.h   All L18
```

Kursstandard: implementasjon

The image shows two side-by-side Emacs buffers. The left buffer, titled 'studreg — emacs-24.3 — 40x22', contains the following code:

```
#include "class_student.h"

#include <iostream>
#include <stdexcept>

using namespace std;

string student::nr(){return studnr;}
```

A blue callout bubble points from the 'student::nr()' line to the right buffer, containing the following text:

**"Namespace operator"
brukes for å referere til
medlemmene "fra utsiden"**

The right buffer, also titled 'studreg — emacs-24.3 — 40x22', contains the header file code:

```
#define CLASS_STUDENT

#include <fstream>
#include <vector>

class student{
    std::vector<std::string> names;
    std::string email;
    std::string studnr;
public:
    student(std::ifstream &src);
    std::string to_string();
    std::string nr();
    static bool valid_studnr(std::string \
s);
};

#endif
```

At the bottom of the right buffer, it says 'All L18'.

Kursstandard: implementasjon

```
studreg — emacs-24.3 — 40x22
File Edit Options Buffers Tools C++ Help
#include "class_student.h"

#include <iostream>
#include <stdexcept>

using namespace std;
string Student::nr() {return studnr;}
string Student::email() {
    return names.front() + '@' + email;
}

student::student(ifstream &file):
    email(), names(), studnr()
{
    string s;
    //...
-UU-:***-F1  class_student.cpp  Top L19
```

Eksplisitt initialisering av
alle medlemmer. C++11-
måten er også lov!

```
studreg — emacs-24.3 — 40x22
File Edit Options Buffers Tools C Help
#ifndef CLASS_STUDENT
#define CLASS_STUDENT

#include <fstream>
#include <vector>

class student{
    std::vector<std::string> names;
    std::string email;
    std::string studnr;
public:
    student(std::ifstream &src);
    std::string to_string();
    std::string nr();
    static bool valid_studnr(std::string \
s);
}

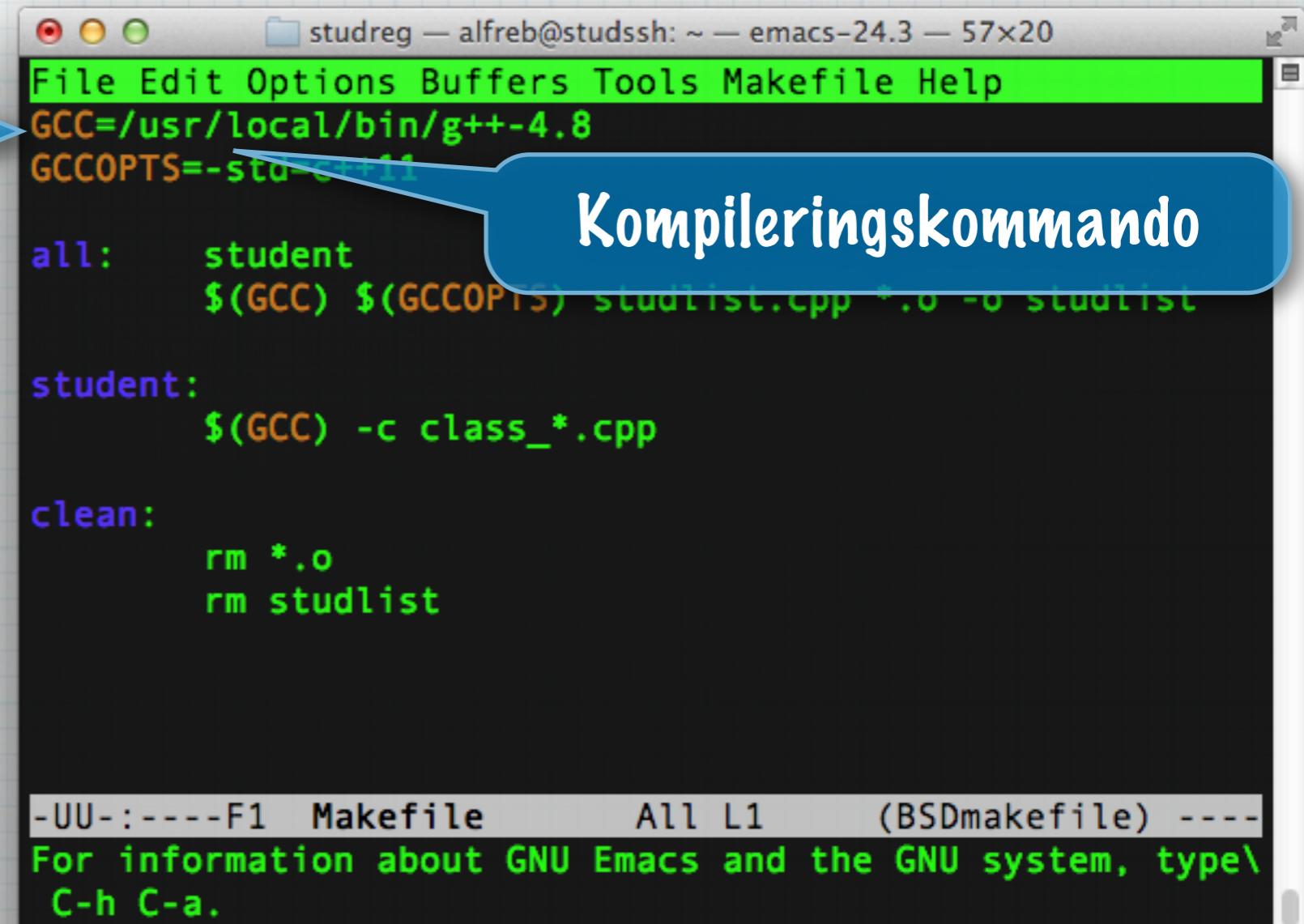
#endif
-UU-:***-F1  class_student.h  All L18
```

Makefile

- * Programmet “Make” fra GNU er et nyttig kompilerings- og installasjonsverktøy
- * I prinsippet et “script”, med shell-kommandoer++
- * Hvis en mappe inneholder en fil “Makefile” vil denne kjøres med kommandoen “make”
- * Makefila består av “merkelapper”, en for hver kompileringsjobb
- * Typiske merkelapper: “all”, “clean”, “configure”, “install”

Makefile: Eksempel

Variabler, nyttig av samme
grunn som alltid.



```
studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20
File Edit Options Buffers Tools Makefile Help
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:      student
          $(GCC) $(CCOPTS) studlist.cpp *.o -o studlist

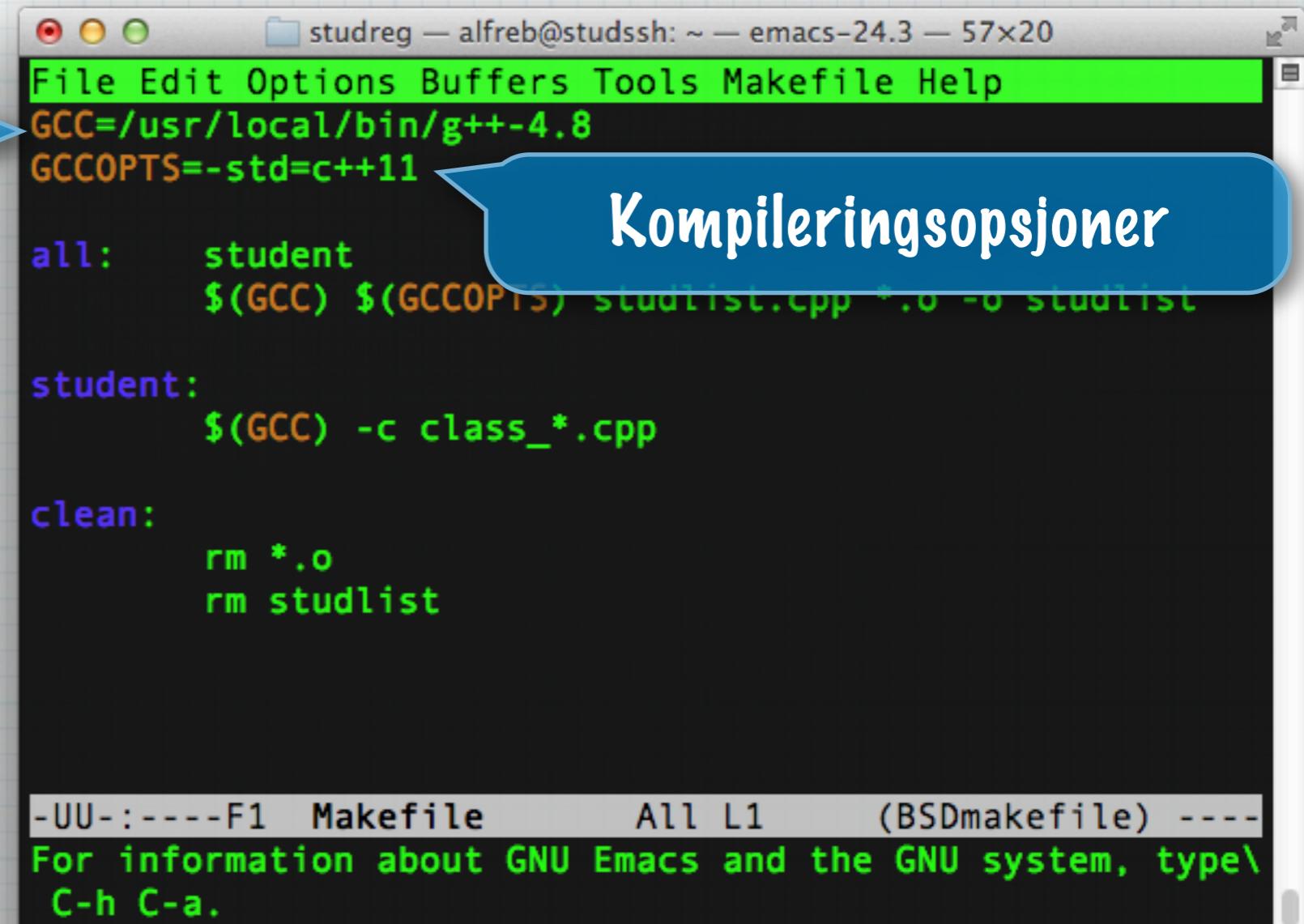
student:
          $(GCC) -c class_*.cpp

clean:
          rm *.o
          rm studlist

-UU-:----F1  Makefile      All L1      (BSDmakefile) ----
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Makefile: Eksempel

Variabler, nyttig av samme
grunn som alltid.



```
studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20
File Edit Options Buffers Tools Makefile Help
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:    student
        $(GCC) $(CCOPTS) studlist.cpp *.o -o studlist

student:
        $(GCC) -c class_*.cpp

clean:
        rm *.o
        rm studlist

-UU-:----F1  Makefile      All L1      (BSDmakefile) ----
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Makefile: Eksempel

Variabler, nyttig av samme
grunn som alltid.

```
studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20
File Edit Options Buffers Tools Makefile Help
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:    student
        $(GCC) $(CCOPTS) studlist.cpp *.o -o studlist

student:
        $(GCC) -c class_.cpp

clean:
        rm *.o
        rm studlist

-UU-:----F1  Makefile      All L1      (BSDmakefile) ----
For information about GNU Emacs and the GNU system, type\
C-h C-a.
```

Brukes slik

Makefile: Eksempel

Variabler, nyttig av samme grunn som alltid.

Merkelapper, for ulike "deljobber"

```
studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20
File Edit Options Buffers Tools Makefile Help
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:    student
        $(GCC) $(CCOPTS) studlist.cpp *.o -o studlist

student:
        $(GCC) -c class_*.cpp

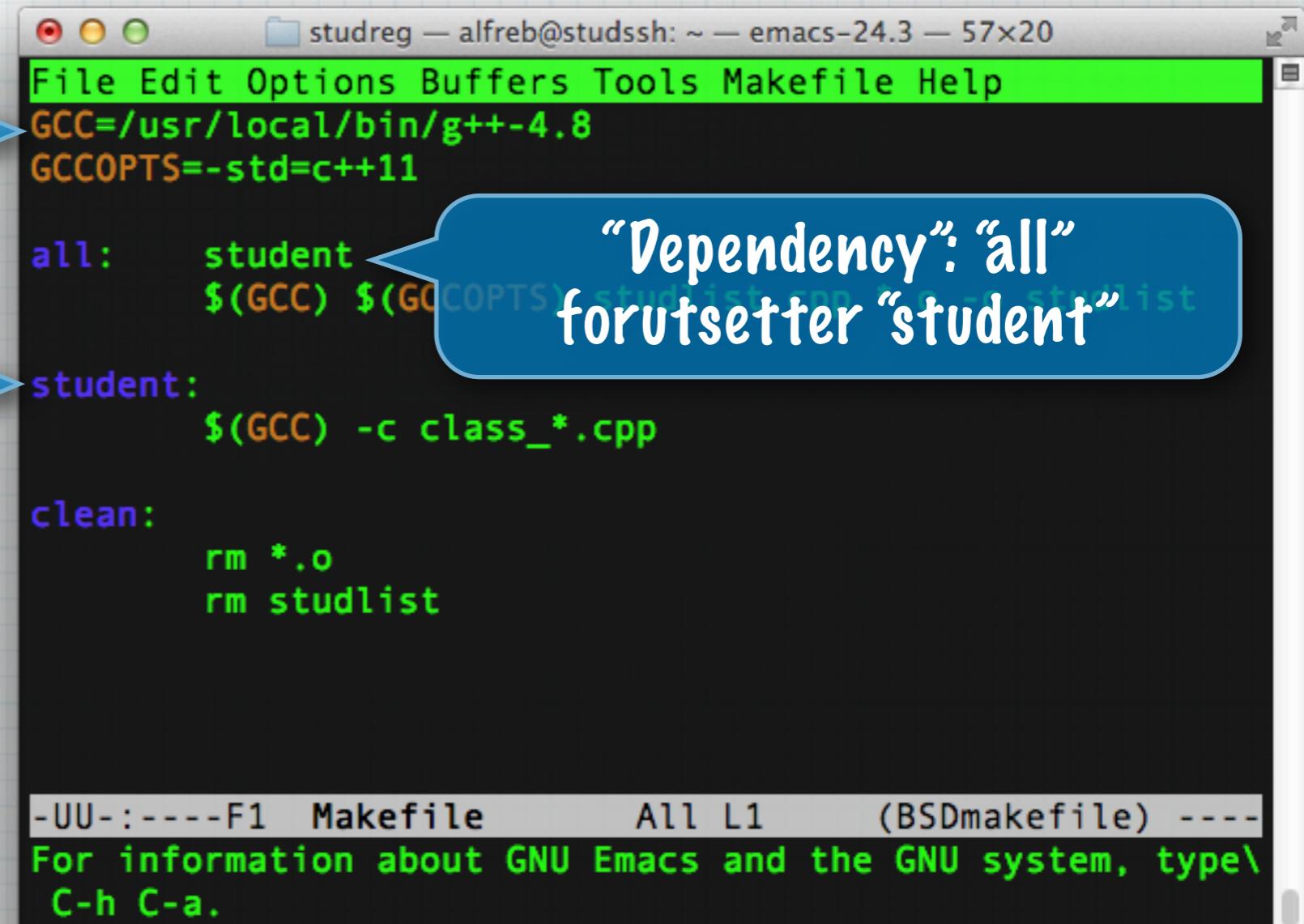
clean:
        rm *.o
        rm studlist

-UU-:----F1  Makefile      All L1      (BSDmakefile) ----
For information about GNU Emacs and the GNU system, type\ C-h C-a.
```

Makefile: Eksempel

Variabler, nyttig av samme grunn som alltid.

Merkelapper, for ulike "deljobber"



The screenshot shows an Emacs window with a Makefile buffer. The window title is "studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20". The Makefile defines variables for the C++ compiler (GCC) and its options (GCCOPTS), and includes targets for "all", "student", and "clean". A tooltip points to the "all" target with the text: "Dependency": "all" forutsetter "student". The bottom status bar displays file information: "-UU- :----F1 Makefile All L1 (BSDmakefile) ----" and a message: "For information about GNU Emacs and the GNU system, type\\ C-h C-a."

```
File Edit Options Buffers Tools Makefile Help
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:    student
        $(GCC) $(GOPTS)

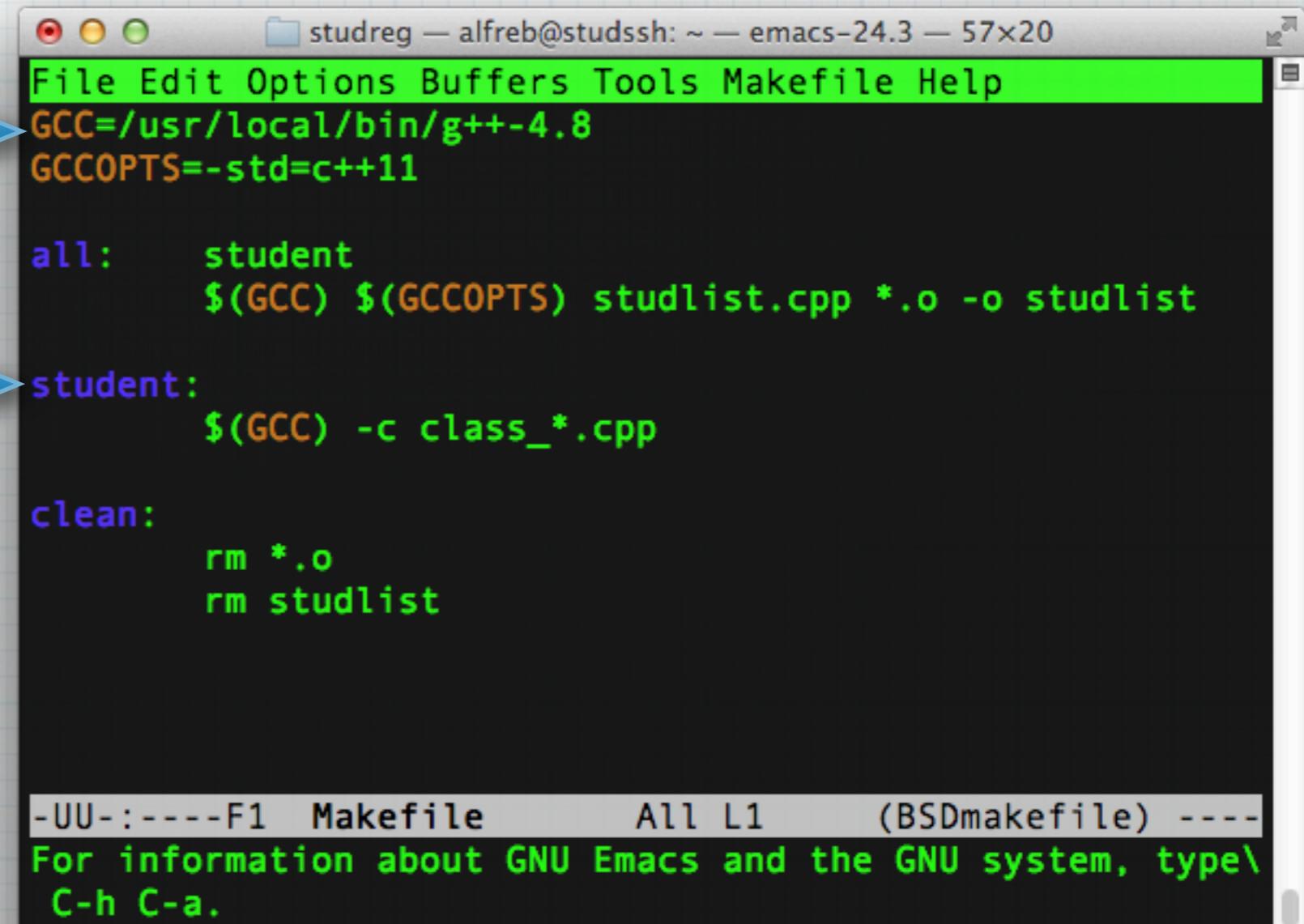
student:
        $(GCC) -c class_*.cpp

clean:
        rm *.o
        rm studlist
```

Makefile: Eksempel

Variabler, nyttig av samme grunn som alltid.

Merkelapper, for ulike "deljobber"



The screenshot shows an Emacs window with a Makefile buffer. The window title bar reads "studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Makefile", and "Help". The Makefile content is as follows:

```
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:    student
        $(GCC) $(CCOPTS) studlist.cpp *.o -o studlist

student:
        $(GCC) -c class_*.cpp

clean:
        rm *.o
        rm studlist
```

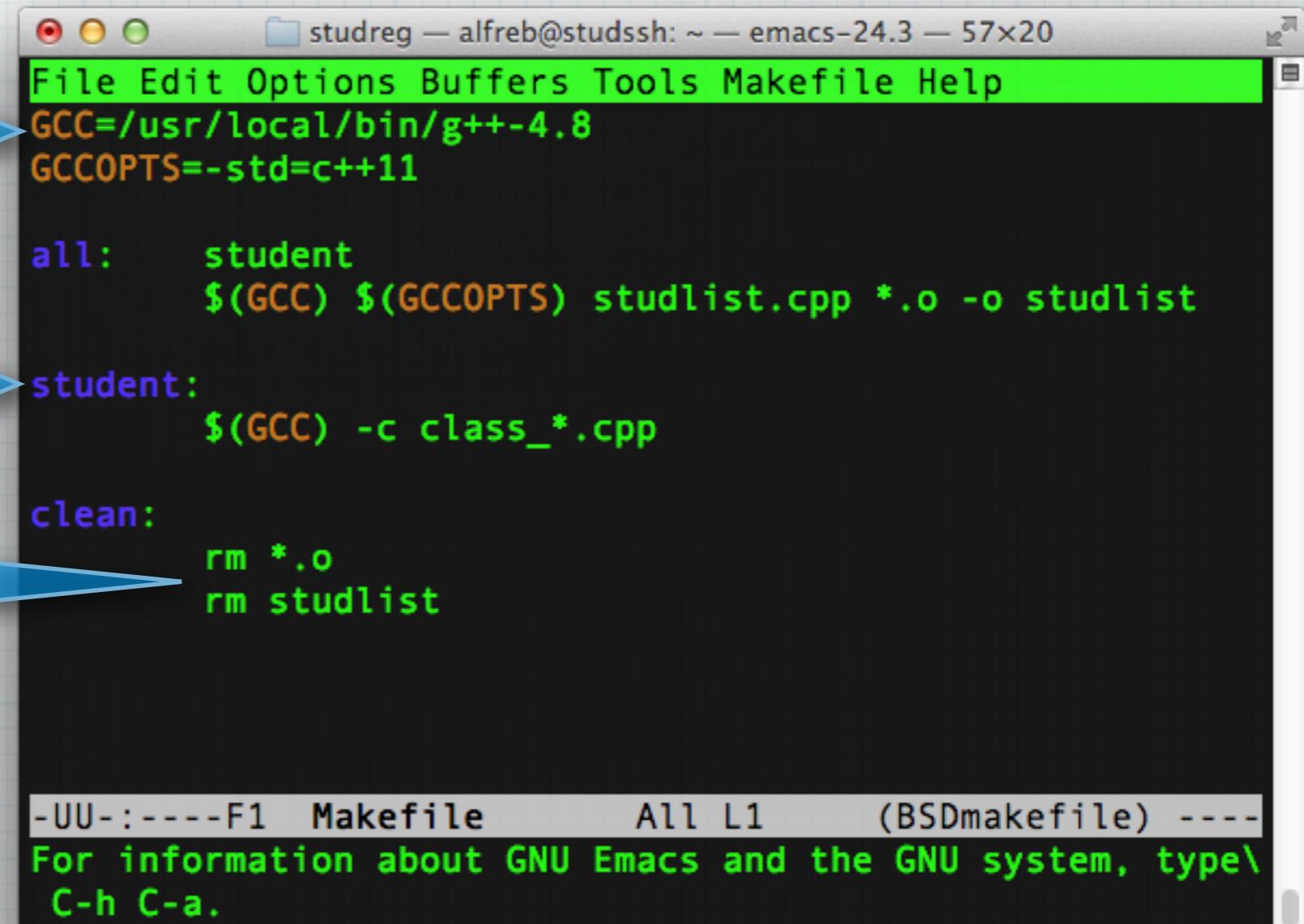
The status bar at the bottom displays "-UU-:----F1 Makefile All L1 (BSDmakefile) ----" and "For information about GNU Emacs and the GNU system, type\\ C-h C-a."

Makefile: Eksempel

Variabler, nyttig av samme grunn som alltid.

Merkelapper, for ulike "deljobber"

Vanlige shell-kommandoer kan brukes fritt



The screenshot shows an Emacs window with a Makefile buffer. The window title is "studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20". The buffer contains the following Makefile code:

```
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:    student
        $(GCC) $(CCOPTS) studlist.cpp *.o -o studlist

student:
        $(GCC) -c class_*.cpp

clean:
        rm *.o
        rm studlist
```

The status bar at the bottom of the Emacs window displays the following information:

```
-UU- :----F1 Makefile      All L1      (BSDmakefile) ----
For information about GNU Emacs and the GNU system, type\
C-h C-a.
```

Makefile: Eksempel

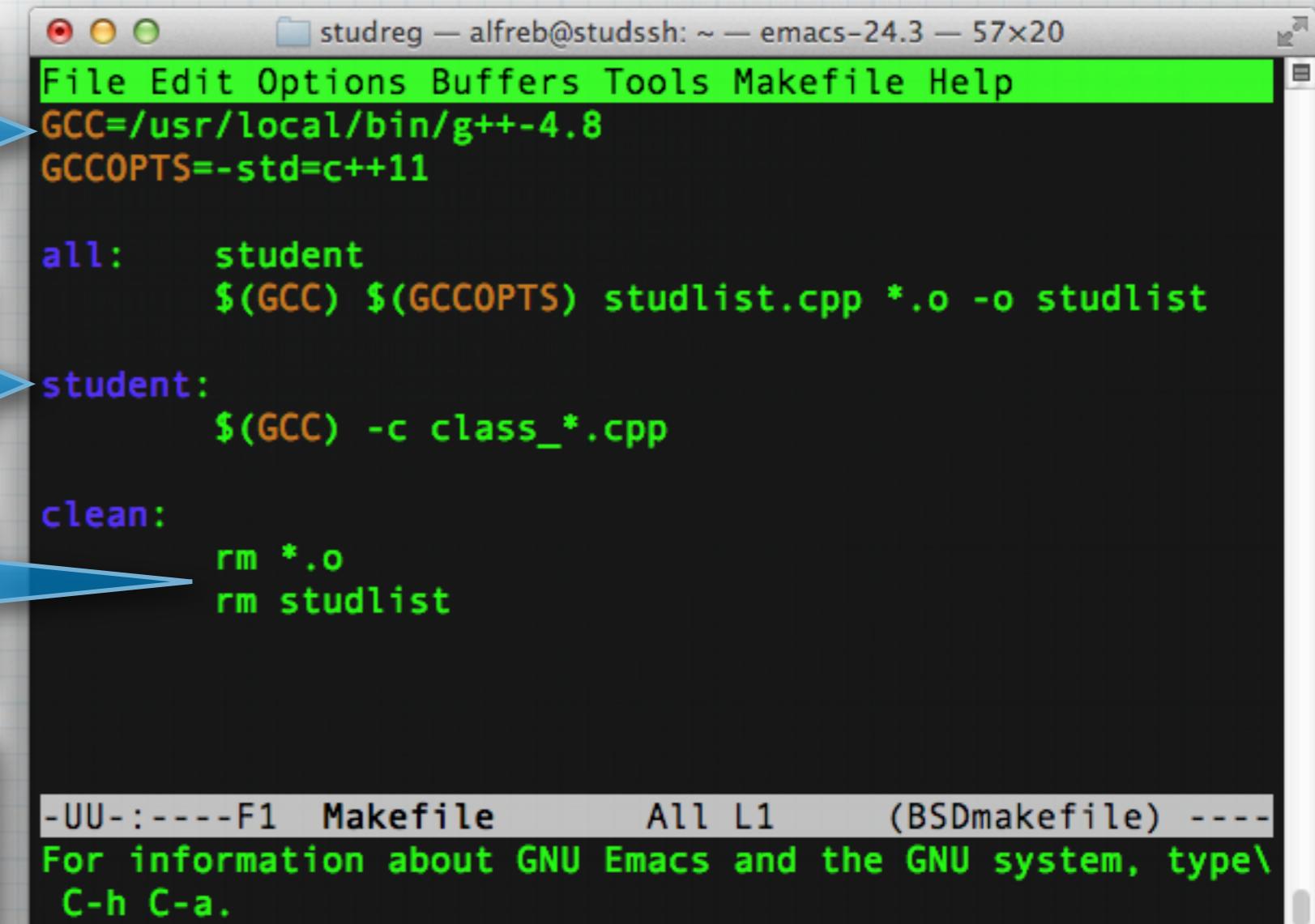
Variabler, nyttig av samme grunn som alltid.

Merkelapper, for ulike "deljobber"

Vanlige shell-kommandoer kan brukes fritt

Makefil skal være med i alle prosjekter form. oblig2

Gullstandard i alle UNIX-systemer



The screenshot shows an Emacs window with a dark theme, displaying a Makefile. The window title bar reads "studreg — alfreb@studssh: ~ — emacs-24.3 — 57x20". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Makefile", and "Help". The Makefile content is as follows:

```
File Edit Options Buffers Tools Makefile Help
GCC=/usr/local/bin/g++-4.8
GCCOPTS=-std=c++11

all:      student
          $(GCC) $(CCOPTS) studlist.cpp *.o -o studlist

student:
          $(GCC) -c class_*.cpp

clean:
          rm *.o
          rm studlist
```

At the bottom of the screen, the Emacs status bar displays: "-UU- :----F1 Makefile All L1 (BSDmakefile) ----", "For information about GNU Emacs and the GNU system, type\\ C-h C-a.", and a vertical scroll bar on the right.

Pause?

Exceptions

- * Exceptions er objekter vi kan kaste tilbake til de som kalte oss, hvis noe galt oppstår.
- * I C har vi "errno.h" som definerer nummer på ulike typer vanlige feil. Disse returneres gjerne som "int'er".
- * Hva skal vi da med exceptions?
- * Slippe utrolig mange "if-statements!"
(mange if == mange feilkilder)
- * Exceptions hopper nemlig nedover på stack, helt til de finner en "catch".
- * Alle steder på stack som hoppes over, måtte ellers hatt "if-setninger".

Exceptions

sp>

```
gcd(0) : throw(err());
```

...

```
gcd(7)
```

```
int i=7...
```

```
gcd(8);
```

```
int i=8;
```

```
gcd(9)
```

```
int i=9;
```

```
gcd(10)
```

```
int x=10; ...
```

```
try{ gcd(11);} catch(err){}
```

```
main: is_prime(10)
```

- * En exception kastes ved å si "throw ..." der ... er et objekt.
- * Stacken "hoppes over"
- * Helt ned til "catch"
- * Og ingen kall imellom trenger å sjekke om det ble returnert riktig
- * ...ser dere noen problemer?

Exceptions

sp>

```
gcd(0) : throw(err());
```

```
...
```

```
gcd(7)
```

```
int i=7...
```

```
gcd(8);
```

```
int* i=new int(8);
```

```
gcd(9)
```

```
int i=9;
```

```
gcd(10)
```

```
int x=10; ...
```

```
try{ gcd(11);} catch(err){}
```

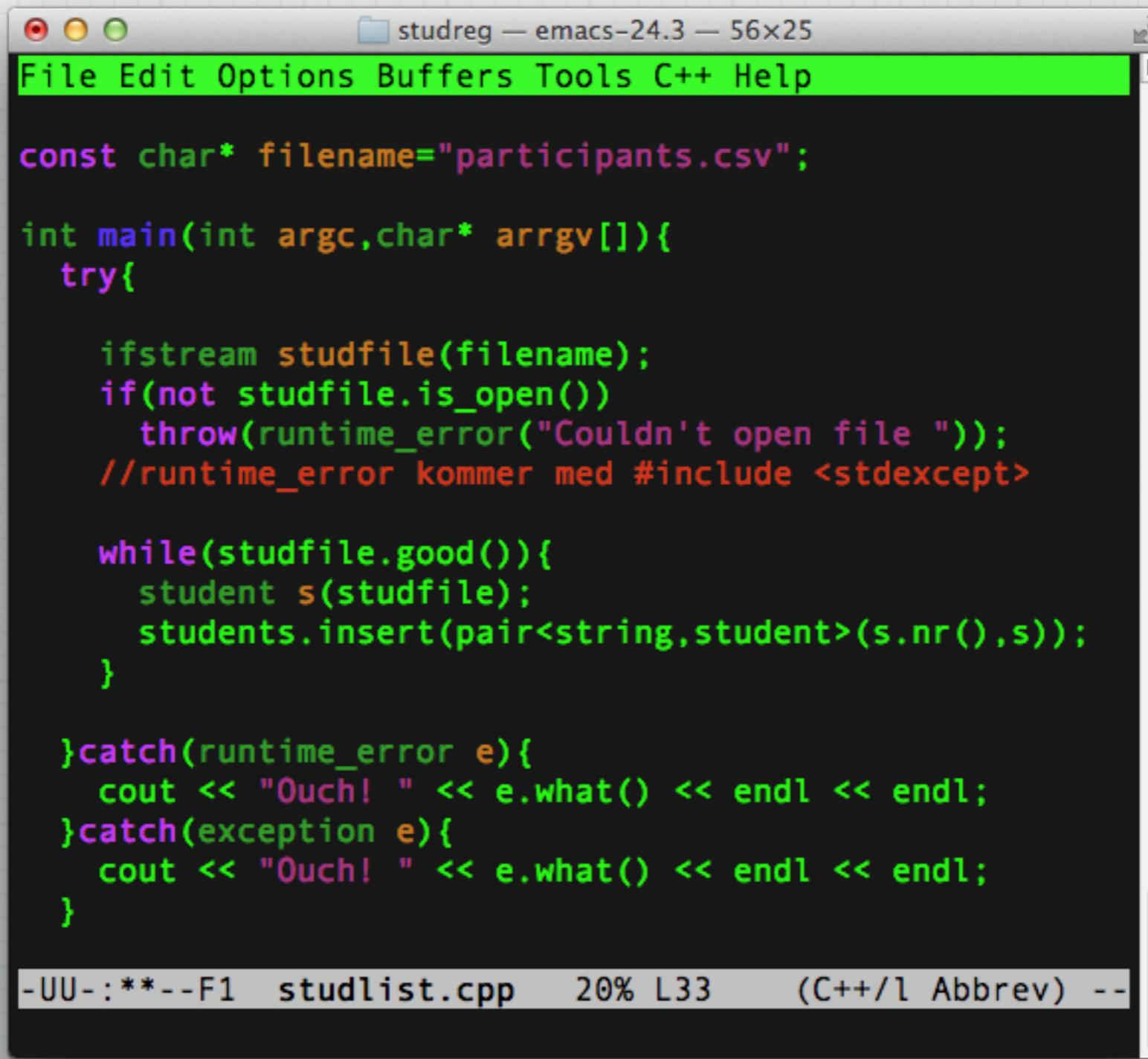
```
main: is_prime(10)
```

- * Hva hvis noe ble allokerert på veien?
- * Det finnes lure løsninger, men inntil videre: Bruk exceptions! (men pass på!)
- * ...Fungerer dette bedre i java?
- * ...Vel - garbage-collector vil rydde opp
- men den må jo rydde opp da.
- * Er det verdt det?
- * JA! Heller få noen feil fra exceptions,
enn å få feil fordi feilhåndteringen er
så innviklet.

Exceptions forts.

- * Vi kan i prinsippet kaste og fange alle typer objekter. Men, vi har noen standarder. (Og - kompilatoren kan ikke vite hva du kommer til å "catche", så den kan ikke sjekke typen)
- * Bruk gjerne `<stdexcept>`: exceptions som tar string som argument til konstruktør.
- * Eller lag egne subklasser av `<exception>` (`stdexcept` arver `exception`). Vent til vi har lært om arv.
- * Alle exceptions har en "string what()" som gir en feilmelding.
- * For alle funksjoner som kan kaste exceptions - bruk `try/catch`. STL oppgir alltid hva som kan kastes.

Exceptions: Eksempel



The screenshot shows an Emacs window titled "studreg — emacs-24.3 — 56×25". The buffer contains the following C++ code:

```
const char* filename="participants.csv";

int main(int argc,char* argv[]){
    try{

        ifstream studfile(filename);
        if(not studfile.is_open())
            throw(runtime_error("Couldn't open file "));
        //runtime_error kommer med #include <stdexcept>

        while(studfile.good()){
            student s(studfile);
            students.insert(pair<string,student>(s.nr(),s));
        }

    }catch(runtime_error e){
        cout << "Ouch! " << e.what() << endl << endl;
    }catch(exception e){
        cout << "Ouch! " << e.what() << endl << endl;
    }

}
```

The status bar at the bottom of the window displays: "-UU- : **--F1 studlist.cpp 20% L33 (C++/l Abbrev) --".

Iteratorer

- * En iterator er en "tryggere peker", beregnet for iterering
 - * Du kan ikke iterere ut av datastrukturen (prøv: `it++` fungerer så langt du vil, men går ikke ut av datastrukturen)
 - * Den kan ikke cast'es til en annen type, selv med C-style cast!
 - * Men, hvis du endrer innholdet i strukturen, og bruker en gammel iterator, vil du få uventet oppførsel. (den kan peke på det gamle elementet)
 - * Hvis elementet er destruert kan du få segfault.
- * Alle stl-containere har iteratorer - og du kan lage dem selv (men litt jobb)
- * Hvordan itererer man over en lenket liste? Vi har prøvd `++...`
- * Jo: `myIterator++` kan overlastes til å gjøre hva som helst:-)

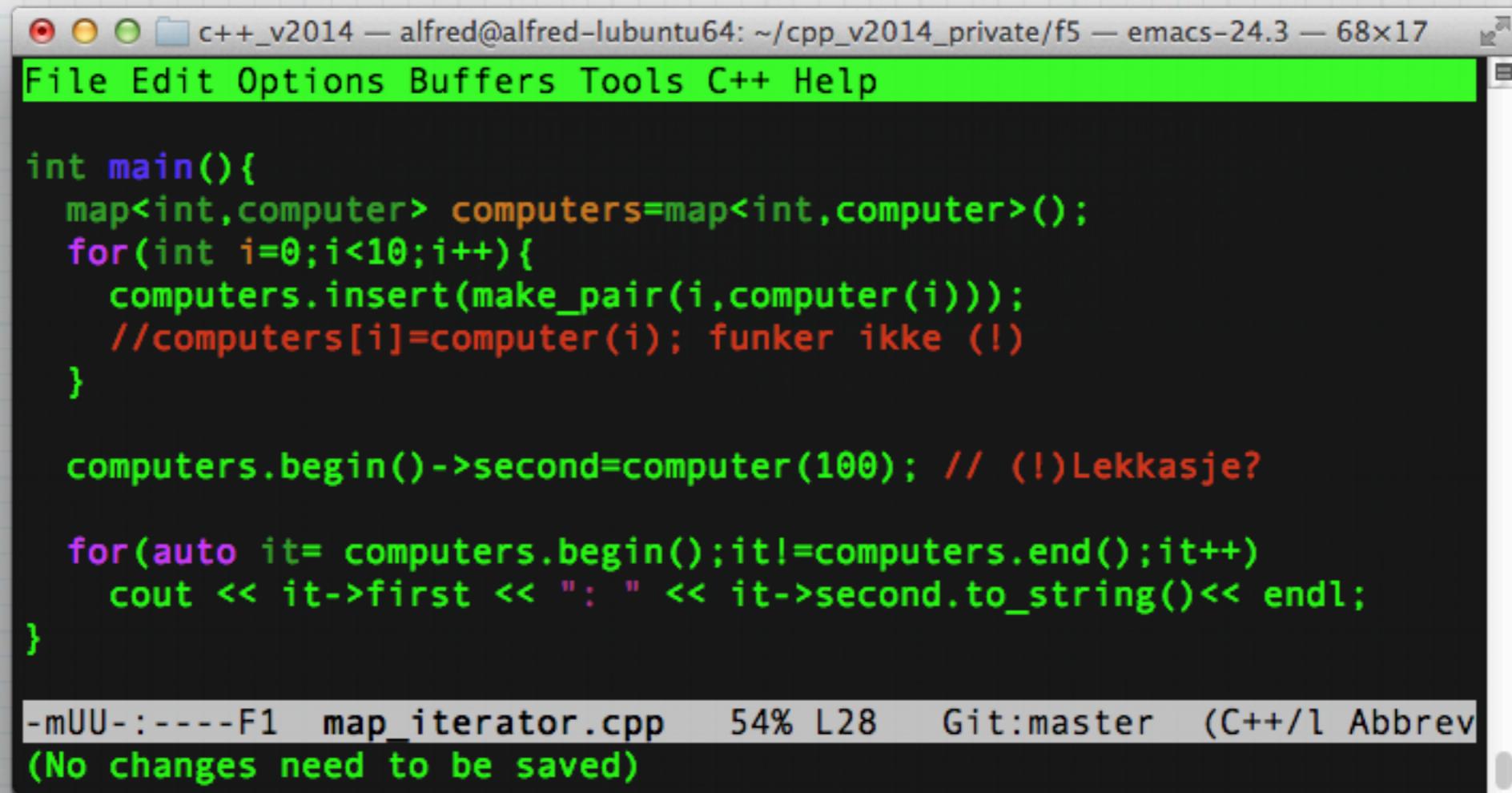
Iteratorer

- * For std::list og bruker du en iterator slik:
`list<computer> computers; //...fyll så i noen maskiner
for(list<computer>::iterator it=computers.begin(); it!=computers.end();it++)
 cout << it->to_string() << endl;`
- * ...litt tungvint. C++11: auto!
- * `for(auto it=computers.begin(); it!=computers.end();it++)
 cout << it->to_string() << endl;`
- * **auto** finner her typen for deg. Det kan den fordi den kan sjekke hva begin() returnerer.
- * Vi bruker :: namespace-operator for å få tilgang til en privat type inni liste-klassen.
- * Legg merke til bruken av ++ og -> **Vi bruker iteratoren som om den var en peker**

Oppklaring: Map og nøkler

- * Jeg nevnte på lab at std::map ikke tar vare på nøklene i en egen datastruktur.
- * Det er riktig - men den tar faktisk vare på nøklene.
 - * ...men vi trenger iteratorer for å finne dem
- * std::map inneholder verdier av std::pair, som er en enkel klasse for par: nøkkel/verdi, "first" og "second"
- * **map<T>::iterator** iteratorer itererer over alle parene
 - * **it->first** gir deg da nøkkelen! (read only)
 - * **it->second** gir deg da verdien (read/write)

Oppklaring: Map og nøkler



The screenshot shows an Emacs window with the following details:

- Title Bar:** c++_v2014 — alfred@alfred-lubuntu64: ~/cpp_v2014_private/f5 — emacs-24.3 — 68x17
- Menu Bar:** File Edit Options Buffers Tools C++ Help
- Code Area:**

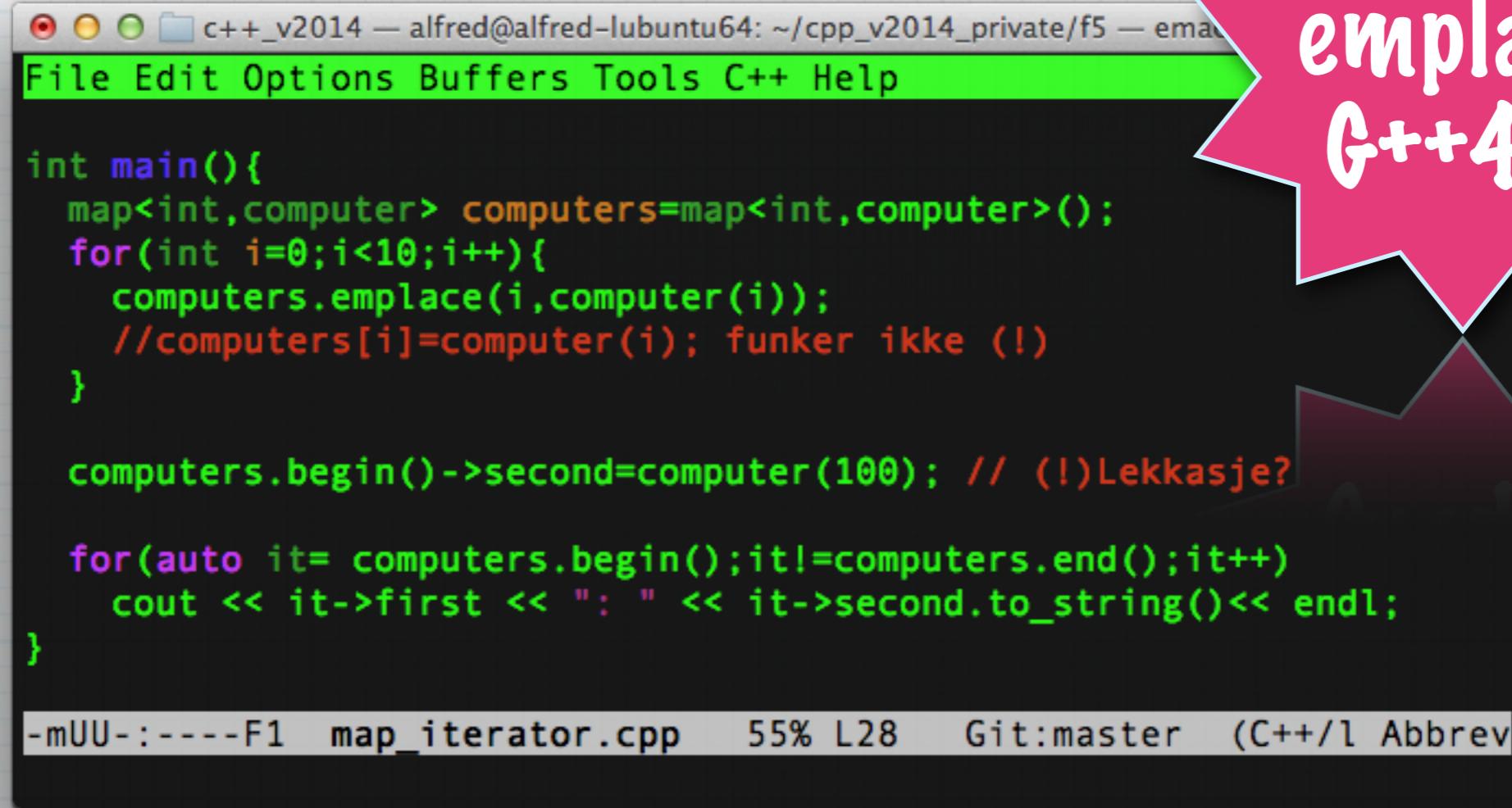
```
int main(){
    map<int,computer> computers=map<int,computer>();
    for(int i=0;i<10;i++){
        computers.insert(make_pair(i,computer(i)));
        //computers[i]=computer(i); funker ikke (!)
    }

    computers.begin()->second=computer(100); // (!)Lekkasje?

    for(auto it= computers.begin();it!=computers.end();it++)
        cout << it->first << ":" << it->second.to_string()<< endl;
}

-mUU-----F1  map_iterator.cpp  54% L28  Git:master  (C++/l Abbrev
(No changes need to be saved)
```
- Status Bar:** -mUU-----F1 map_iterator.cpp 54% L28 Git:master (C++/l Abbrev
(No changes need to be saved)

Oppklaring: Map og nøkler



```
c++_v2014 — alfred@alfred-lubuntu64: ~/cpp_v2014_private/f5 — emacs
File Edit Options Buffers Tools C++ Help

int main(){
    map<int,computer> computers=map<int,computer>();
    for(int i=0;i<10;i++){
        computers.emplace(i,computer(i));
        //computers[i]=computer(i); funker ikke (!)
    }

    computers.begin()->second=computer(100); // (!)Lekkasje?

    for(auto it= computers.begin();it!=computers.end();it++)
        cout << it->first << ":" << it->second.to_string()<< endl;
}

-mUU-:----F1  map_iterator.cpp  55% L28  Git:master  (C++/l Abbrev
```

emplace
C++4.8

Demo?

exception_leak.cpp
map_iterator.cpp
studlist.cpp
Makefile