

# Effektiv Kode med C og C++

---

Forelesning 5, vår 2015  
Alfred Bratterud



# Agenda:

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



# Litt repetisjon

---



# konstruktører i klasser

- \* Konstruktøren har som jobb å initialisere alle medlemmer - ved å kalle deres konstruktører.
- \* **OBS:** Lager du en konstruktør selv, uten argumenter, mister du den «implisitte» default-konstruktøren som lages for deg
- \* I alle egne konstruktører må alle medlemmer initialiseres manuelt - evt. ved å kalle default-konstruktør.

- \* Man må da initialisere slik:

```
class student{  
    int nr;  
    int nr = 0; // OK i C++11  
    int birth_year{1979} // OK i C++11  
    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 / evt. andre ressurser klassen har satt av.
- \* Destruktoeren til en klasse "myClass" heter " ~myClass()"
- \* Vi kan lage egne, som vanlig, men alle objekter har en "default destructor"
- \* ....Den kaller destructoren til medlemmene. Default destructor for peker?
- \* Hva vil vi typisk gjøre i en "destructor"?
  - \* Frigjøre alt klassen allokererte med new!
  - \* Evt. lukke filer!
- \* Fantastisk garanti: Destructor kalles alltid når variabel går ut av skop.
- \* Hvorfor har vi ikke destructorer i java/php?
  - \* Alle problemene i C++ skyldes pekere ;-)



# Interface v.s. Implementation

- \* “**Interfacet**” 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” (class\_student.hpp)
- \* “**Implementasjonen**” ligger gjerne i en annen fil (class\_student.cpp), som “inkluderer” header-fila
- \* Fordi vi da kan «interface»/«snakke» med ferdig kompilerte klasser, kun ved å kjenne til header fila
- \* Skjer via «linking» mellom din binærfil og den ferdig kompilerte klassen
  - \* Gjøres av «linkeren» (GNU ld i Linux-vm'en)
  - \* Kan gjøres «Statisk» eller «dynamisk» - til og med «run time»
  - \* I Windows: **.dll**'er er delte, ferdigkompilerte biblioteker. I Linux: **.so**
- \* **G++ kompilerer \*og\* linker. For å kun kompilere - bruk `g++ -c myfile.cpp`. Det genererer en kompilert objektfil **myfile.o** som kan linkes med andre filer.**



# Kursstandard: header

“Include guard”:  
En macro som hindrer  
multippel inkludering

“Include guard”

```
#ifndef STUDENT_HPP
#define STUDENT_HPP

#include <fstream>
#include <vector>

class Student{
    std::vector<std::string> names_;
    std::string email_{"N/A"};
    std::string studnr_{"N/A"};

    static int current_nr_;
public:

    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();

    /** Get student number. */
    std::string nr();

    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};

#endif
```



# Kursstandard: header

“Include guard”:  
En macro som hindrer  
multippel inkludering

Vi trenger kanskje «PI» eller  
«Card» flere steder i  
programmet - men klasser  
og variabler kan ikke  
deklarerer flere ganger!  
(Funksjonssignaturer går,  
men ikke kropp.)

“Include guard”

```
#ifndef STUDENT_HPP
#define STUDENT_HPP

#include <fstream>
#include <vector>

class Student{
    std::vector<std::string> names_;
    std::string email_{"N/A"};
    std::string studnr_{"N/A"};

    static int current_nr_;
public:

    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();

    /** Get student number. */
    std::string nr();

    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};

#endif
```



# Kursstandard: header

“Include guard”:  
En macro som hindrer  
multippel inkludering

Ingen “using namespace std;”

“Include guard”

```
#ifndef STUDENT_HPP
#define STUDENT_HPP

#include <fstream>
#include <vector>

class Student{
    std::vector<std::string> names_;
    std::string email_{"N/A"};
    std::string studnr_{"N/A"};

    static int current_nr_;
public:

    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();

    /** Get student number. */
    std::string nr();

    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};

#endif
```



# 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”

```
#ifndef STUDENT_HPP
#define STUDENT_HPP

#include <fstream>
#include <vector>

class Student{
    std::vector<std::string> names_;
    std::string email_{"N/A"};
    std::string studnr_{"N/A"};

    static int current_nr_;
public:

    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();

    /** Get student number. */
    std::string nr();

    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};

#endif
```



# Header + Implementasjon

```
#include "student.hpp"

// Not included in header - why?
#include <iostream>

using namespace std;

Student::Student(initializer_list<string> names,
                 string mail) :
    names_{names}, email_{mail}
{
    cout << "Constructing student "
          << *names.begin() << endl;
}

Student::Student(initializer_list<string> names) :
    Student{names, "N/A"} {}

string Student::str() {
    string name = "";
    for(auto n : names_)
        name += " " + n;
    return name + " <" + email_ + ">";
}
```

```
#ifndef STUDENT_HPP
#define STUDENT_HPP

#include <fstream>
#include <vector>

class Student {
    std::vector<std::string> names_;
    std::string email_{"N/A"};
    std::string studnr_{"N/A"};

    static int current_nr_;
public:

    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();

    /** Get student number. */
    std::string nr();

    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};

#endif
```



# Header + Implementasjon

```
#include "student.hpp"

// Not included in header - why?
#include <iostream>

using namespace std;

Student::Student(initializer_list<string> names,
                 string mail) :
    names_{names}, email_{mail}
{
    cout << "Constructing student "
          << *names.begin() << endl;
}

Student::Student(initializer_list<string> names) :
    Student{names, "N/A"} {}

string Student::str() {
    string name = "";
    for(auto n : names_)
        name += " " + n;
    return name + " <" + email_ + ">";
}
```

```
#ifndef STUDENT_HPP
#define STUDENT_HPP
```

OK i implementasjonen!  
Hvorfor?

```
#include <string>
#include <vector>

class Student {
    std::vector<std::string> names_;
    std::string email_{ "N/A" };
    std::string studnr_{ "N/A" };

    static int current_nr_;
public:

    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();

    /** Get student number. */
    std::string nr();

    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};

#endif
```



# Header + Implementasjon

```
#include "student.hpp"

// Not included in header - why?
#include <iostream>

using namespace std;
```

```
Student::Student(initializer_list<string> names,
                  string mail) :
    names_(names), email_{mail}
{
```

“Namespace operator”  
brukes for å referere til  
medlemmene “fra utsiden”

```
    cout << "Constructing Student " << names_.size() << endl;
}

Student::str() const {
    Student{names, "N/A"}}

string Student::str(){
    string name="";
    for(auto n : names_)
        name+=" "+n;
    return name+" <"+email_+">";
}
```

```
#ifndef STUDENT_HPP
#define STUDENT_HPP
```

OK i implementasjonen!  
Hvorfor?

```
#include <string>
#include <vector>

class Student{
    std::vector<std::string> names_;
    std::string email_{"N/A"};
    std::string studnr_{"N/A"};
```

```
    static int current_nr_;
public:
```

```
    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();
```

```
    /** Get student number. */
    std::string nr();
```

```
    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};
```

```
#endif
```



# Header + Implementasjon

```
#include "student.hpp"

// Not included in header - why?
#include <iostream>

using namespace std;

Student::Student(initializer_list<string> names,
                 string mail) :
    names_{names}, email_{mail}
{
    cout << "Constructing student "
          << *names.begin() << endl;
}

Student::Student(initializer_list<string> names) :
    Student{names, ""} {}

string Student::str() const {
    string name = "";
    for(auto n : names_)
        name += " " + n;
    return name + " <" + email_ + ">";
}
```

Eksplisitt initialisering av  
alle medlemmer.  
(...glemt noe?)

```
#ifndef STUDENT_HPP
#define STUDENT_HPP
```

OK i implementasjonen!  
Hvorfor?

```
#include <string>
#include <vector>

class Student{
    std::vector<std::string> names_;
    std::string email_{"N/A"};
    std::string studnr_{"N/A"};

    static int current_nr_;
public:

    /** Construct with names only. Email will be N/A */
    Student(std::initializer_list<std::string> names);
    Student(std::initializer_list<std::string> names,
            std::string mail);
    std::string str();

    /** Get student number. */
    std::string nr();

    static bool valid_studnr(std::string);
    static std::string generate_studnr();
};

#endif
```

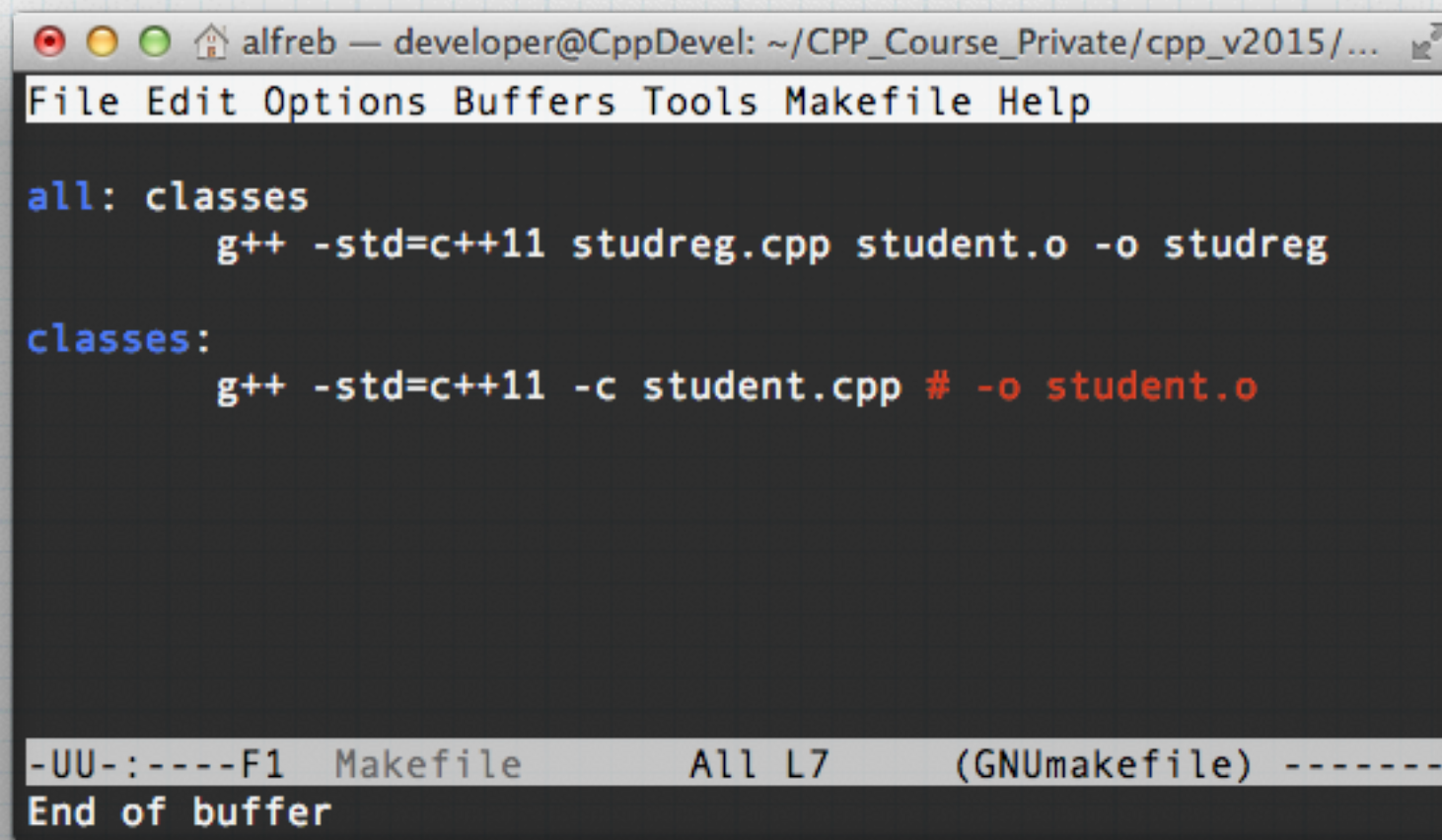


# Makefile

- \* 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"
- \* Finnes alternativer; cmake, nmake etc. GNU Make skal brukes i kurset - oblig2 vil kreve make-fil.



# Makefile: Minimal og bedre enn ingenting



The screenshot shows a window titled "alfreb — developer@CppDevel: ~/CPP\_Course\_Private/cpp\_v2015/...". The window contains a Makefile with the following content:

```
File Edit Options Buffers Tools Makefile Help

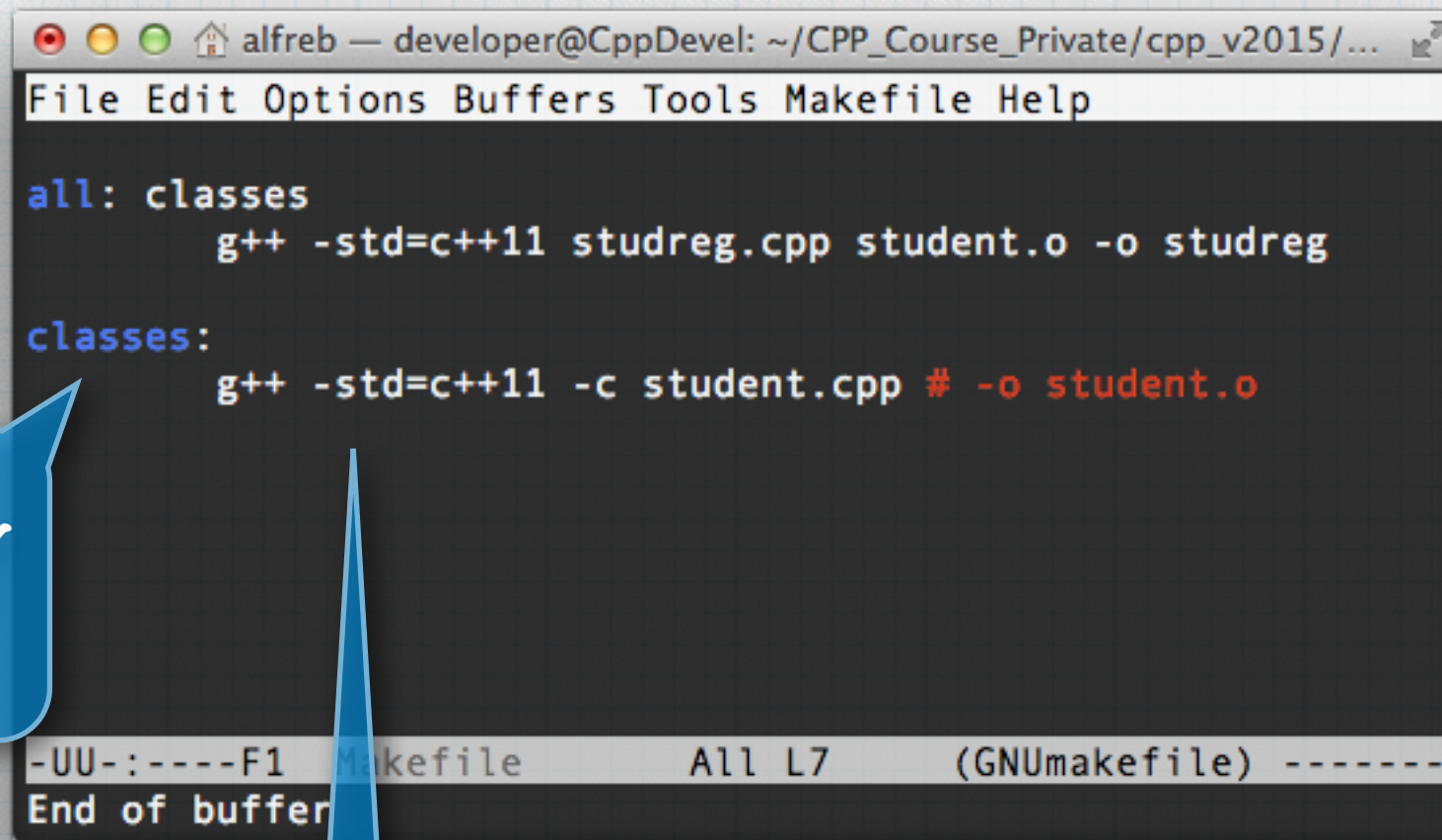
all: classes
    g++ -std=c++11 studreg.cpp student.o -o studreg

classes:
    g++ -std=c++11 -c student.cpp # -o student.o

-UU-:----F1 Makefile      All L7      (GNUmakefile) -----
End of buffer
```



# Makefile: Minimal og bedre enn ingenting



```
File Edit Options Buffers Tools Makefile Help

all: classes
    g++ -std=c++11 studreg.cpp student.o -o studreg

classes:
    g++ -std=c++11 -c student.cpp # -o student.o

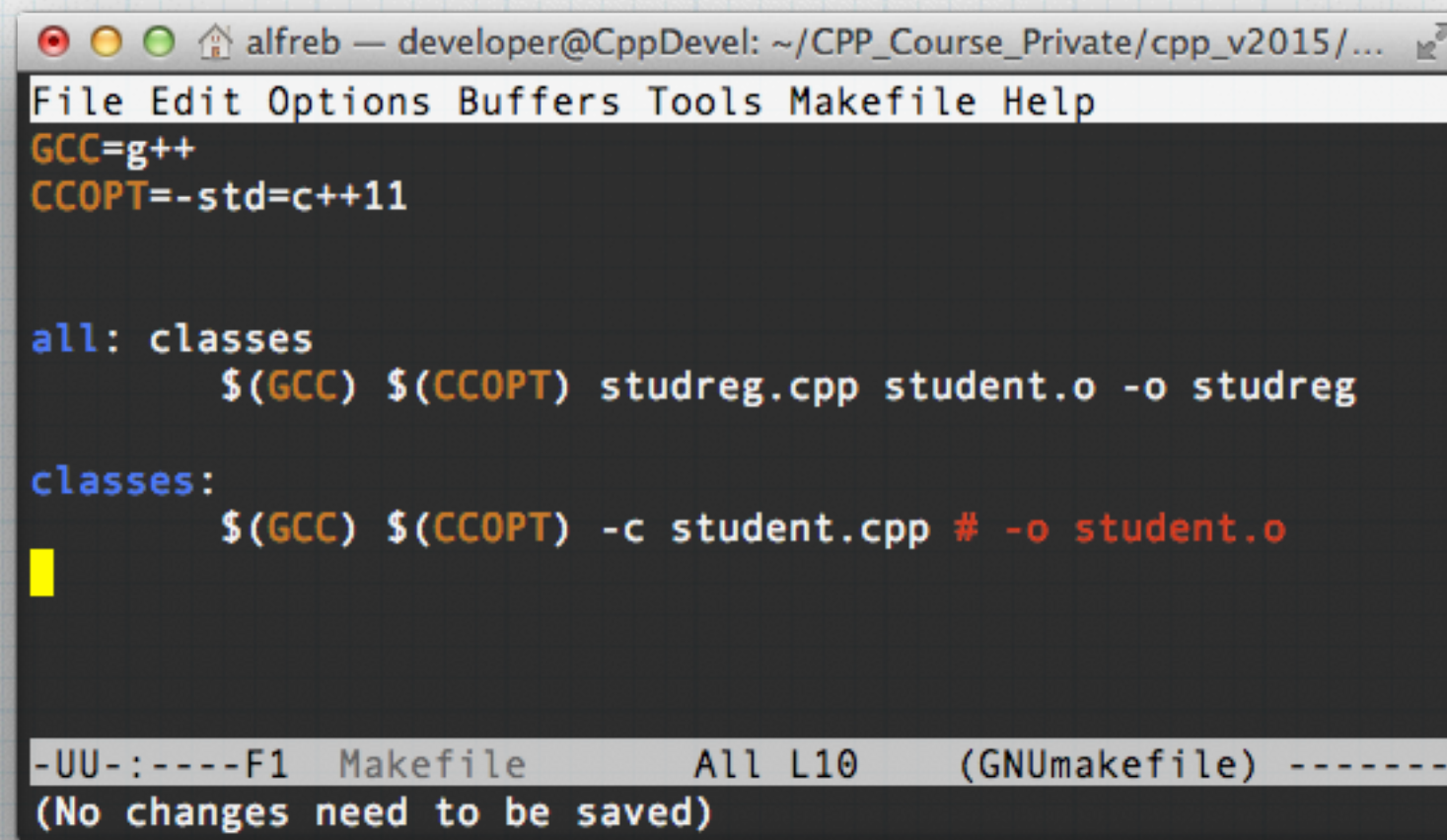
-UU-:----F1 Makefile      All L7      (GNUmakefile) -----
End of buffer
```

Merkelapper, for ulike "deljobber"

Innhold kan være en vanlig shell-kommando



# Makefile: Variabler gjør fila «DRY»



```
alfreb — developer@CppDevel: ~/CPP_Course_Private/cpp_v2015/...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

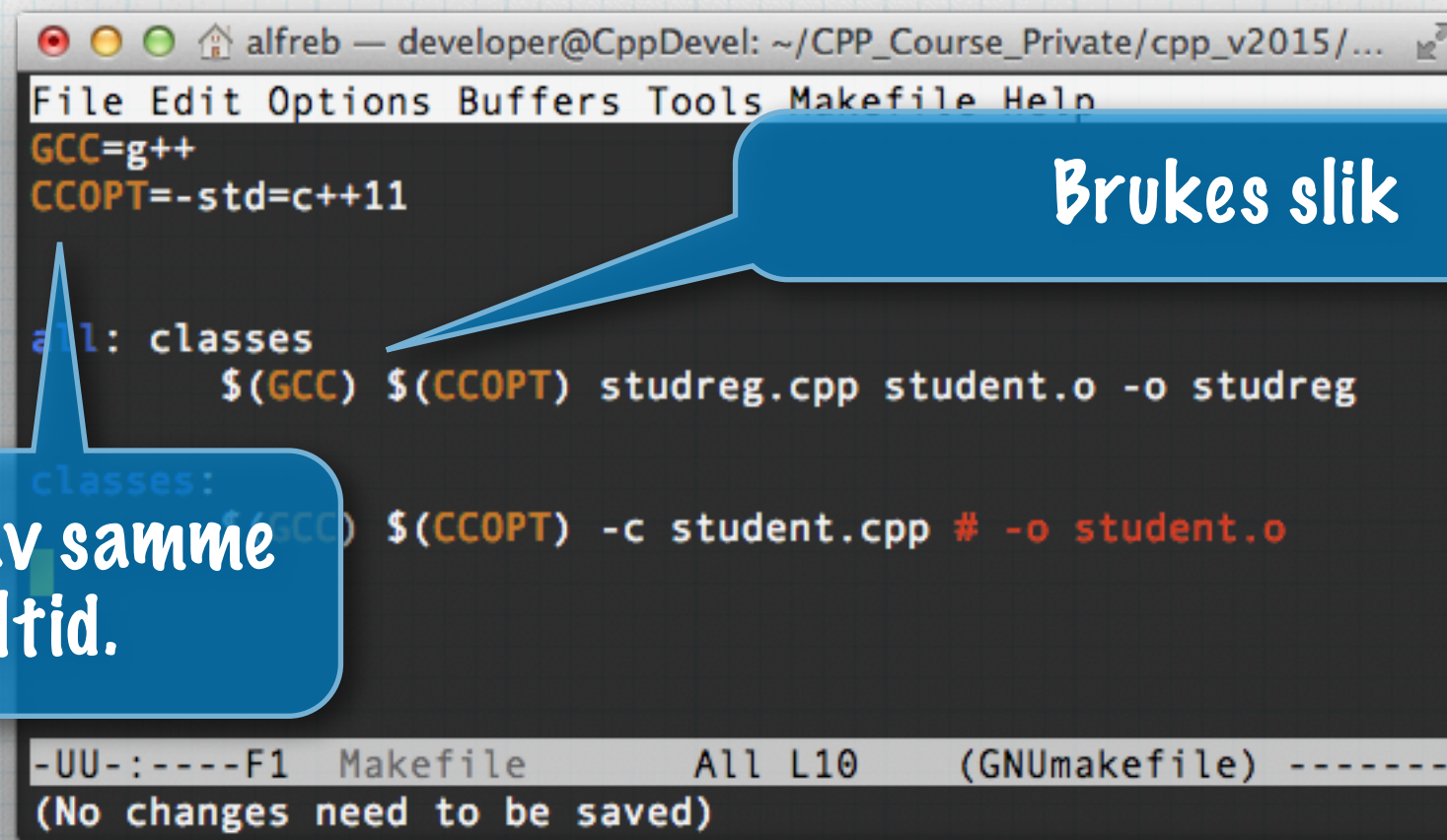
all: classes
    $(GCC) $(CCOPT) studreg.cpp student.o -o studreg

classes:
    $(GCC) $(CCOPT) -c student.cpp # -o student.o

-UU-:----F1 Makefile      All L10      (GNUmakefile) -----
(No changes need to be saved)
```



# Makefile: Variabler gjør fila «DRY»



```
alfreb — developer@CppDevel: ~/CPP_Course_Private/cpp_v2015/...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

all: classes
    $(GCC) $(CCOPT) studreg.cpp student.o -o studreg

classes:
    $(GCC) $(CCOPT) -c student.cpp # -o student.o

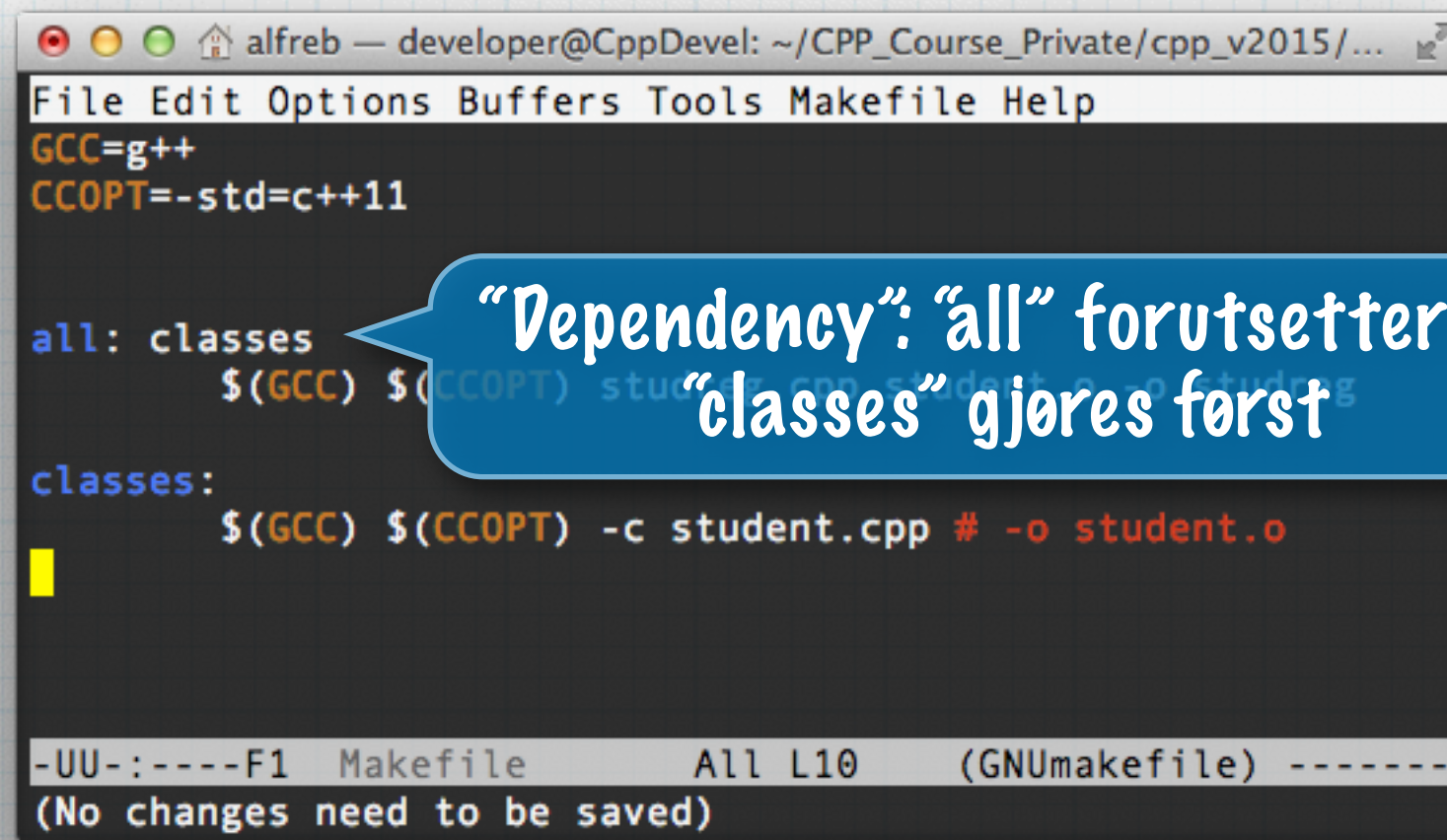
-UU-:----F1 Makefile      All L10      (GNUmakefile) -----
(No changes need to be saved)
```

Brukes slik

Variabler, nyttig av samme grunn som alltid.



# Makefile: Variabler gjør fila «DRY»



```
alfreb — developer@CppDevel: ~/CPP_Course_Private/cpp_v2015/...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

all: classes
    $(GCC) $(CCOPT) student.cpp -o student.o

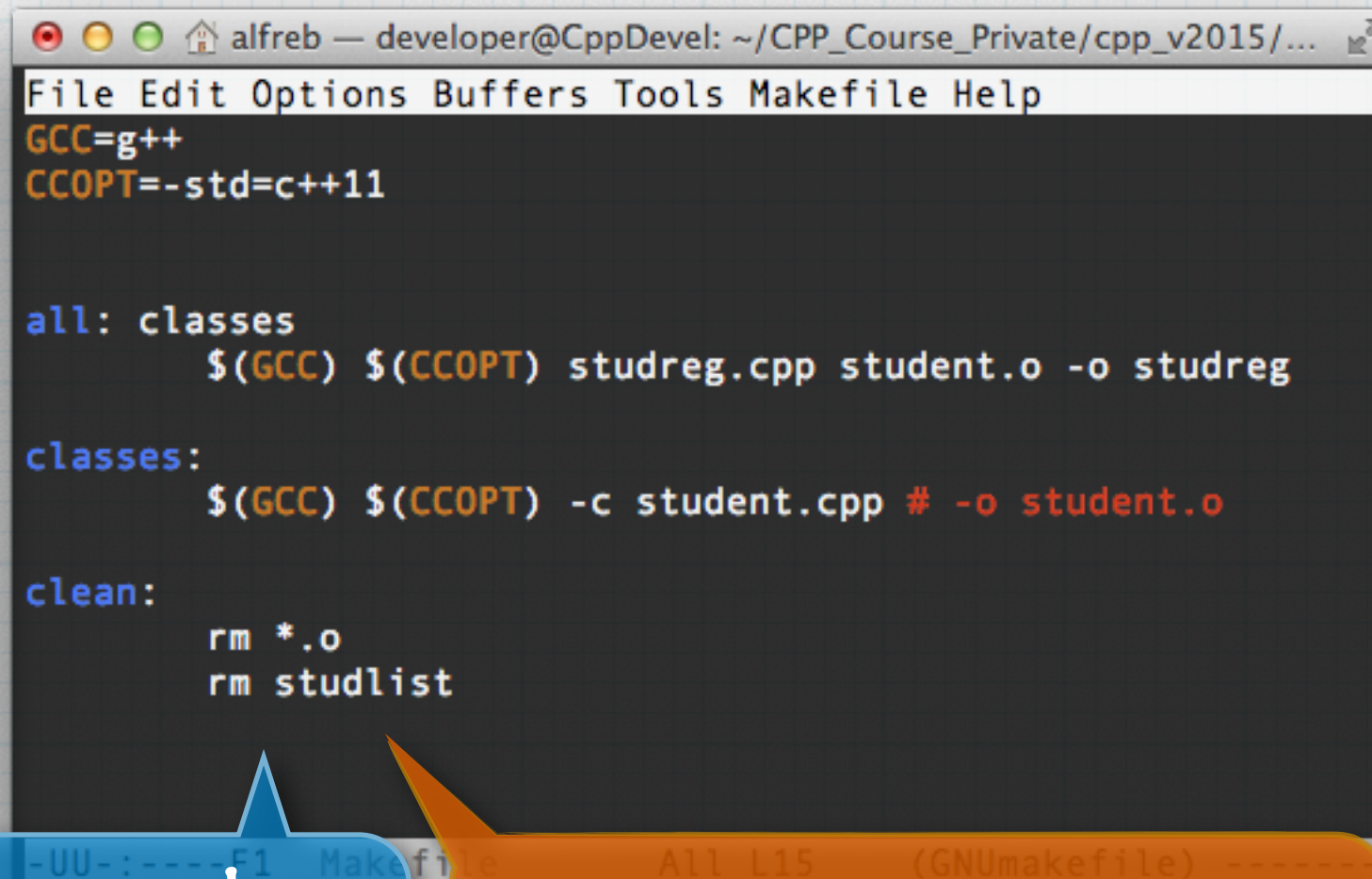
classes:
    $(GCC) $(CCOPT) -c student.cpp # -o student.o

-UU-:----F1 Makefile      All L10      (GNUmakefile) -----
(No changes need to be saved)
```

“Dependency”: “all” forutsetter at  
“classes” gjøres først



# Makefile: Eksempel



```
alfreb — developer@CppDevel: ~/CPP_Course_Private/cpp_v2015/...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

all: classes
    $(GCC) $(CCOPT) studreg.cpp student.o -o studreg

classes:
    $(GCC) $(CCOPT) -c student.cpp # -o student.o

clean:
    rm *.o
    rm studlist
```

Vanlige shell-kommandoer  
kan brukes fritt - med  
wildcards

Men hvis en kommando feiler,  
stopper Make...



# Makefile: Eksempel

```
alfreb — developer@CppDevel: ~/CPP_Course_Private/cpp_v2015/...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

all: classes
    $(GCC) $(CCOPT) studreg.cpp student.o -o studreg

classes:
    $(GCC) $(CCOPT) -c student.cpp # -o student.o

clean:
    $(RM) *.o
    $(RM) studlist
```

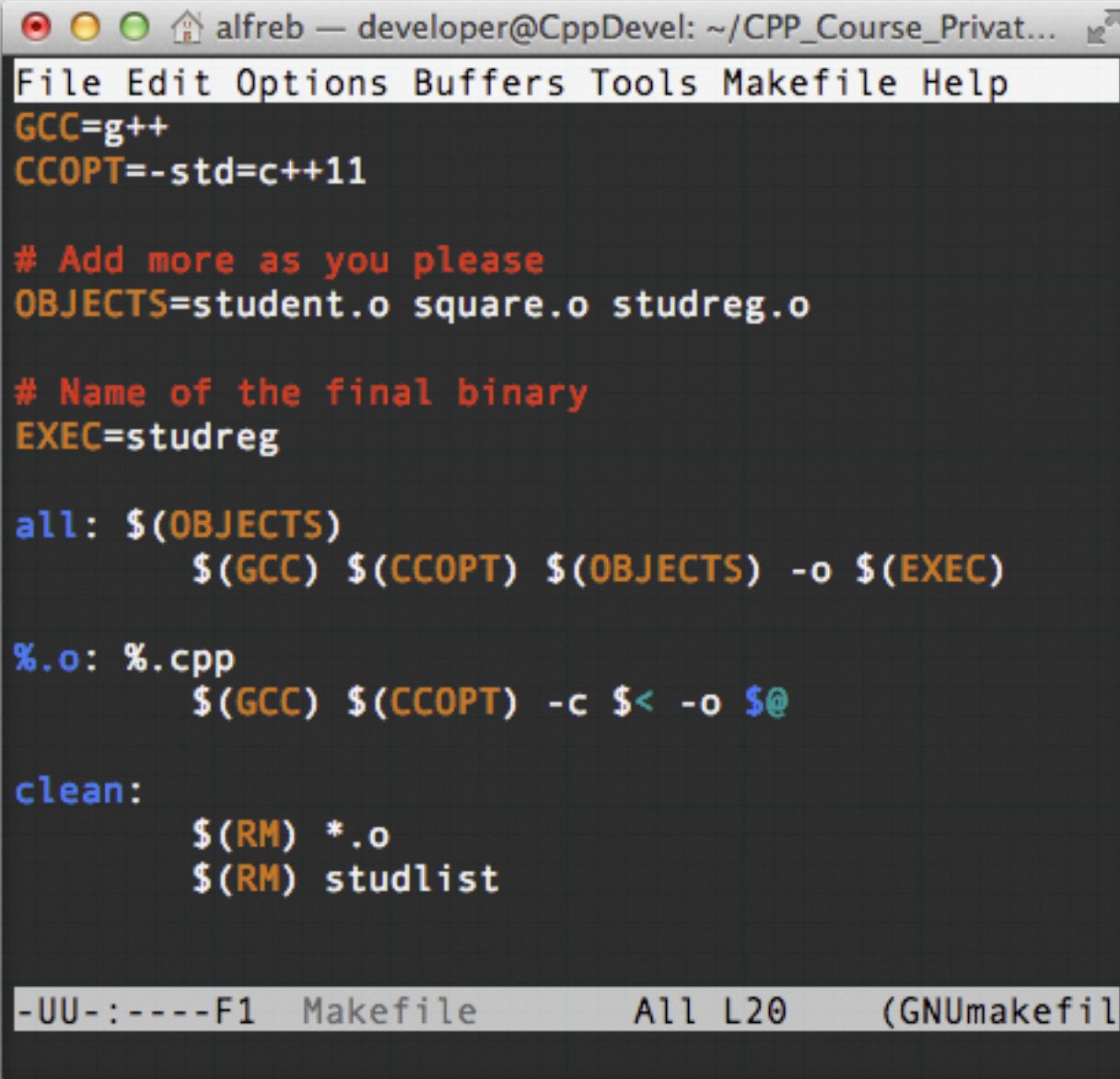
Vanlige shell-kommandoer  
kan brukes fritt - med  
wildcards

Finnes «implicit variables»  
for flere av dem\*

\* [https://www.gnu.org/software/make/manual/html\\_node/Implicit-Variables.html](https://www.gnu.org/software/make/manual/html_node/Implicit-Variables.html)



# Makefile: Fullstendig

A screenshot of a terminal window with a dark background and light-colored text. The window title bar shows 'alfreb — developer@CppDevel: ~/CPP\_Course\_Privat...'. The terminal displays a Makefile with various variables and rules. The variables are: GCC=g++, CCOPT=-std=c++11, OBJECTS=student.o square.o studreg.o, and EXEC=studreg. The rules are: 'all' which depends on \$(OBJECTS) and uses \$(GCC) \$(CCOPT) to compile them into \$(EXEC); a pattern rule for %.o from %.cpp using \$(GCC) \$(CCOPT) -c; and a 'clean' rule that removes \*.o and studlist. The status bar at the bottom shows '-UU-:----F1 Makefile All L20 (GNUmakefil'.

```
alfreb — developer@CppDevel: ~/CPP_Course_Privat...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

# Add more as you please
OBJECTS=student.o square.o studreg.o

# Name of the final binary
EXEC=studreg

all: $(OBJECTS)
    $(GCC) $(CCOPT) $(OBJECTS) -o $(EXEC)

%.o: %.cpp
    $(GCC) $(CCOPT) -c $< -o $@

clean:
    $(RM) *.o
    $(RM) studlist

-UU-:----F1 Makefile All L20 (GNUmakefil
```



# Makefile: Fullstendig

Liste av objektfiler vi skal bygge  
(hver for seg)

Hele listen som «dependency»

```
alfreb — developer@CppDevel: ~/CPP_Course_Privat...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

# Add more as you please
OBJECTS=student.o square.o studreg.o

# Name of the final binary
EXEC=studreg

all: $(OBJECTS)
    $(GCC) $(CCOPT) $(OBJECTS) -o $(EXEC)

%.o: %.cpp
    $(GCC) $(CCOPT) -c $< -o $@

clean:
    $(RM) *.o
    $(RM) studlist

-UU-:----F1 Makefile All L20 (GNUmakefil
```



# Makefile: Fullstendig

Liste av objektfiler vi skal bygge  
(hver for seg)

Vanlig variabel (her, binærfile)

```
alfreb — developer@CppDevel: ~/CPP_Course_Privat...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

# Add more as you please
OBJECTS=student.o square.o studreg.o

# Name of the final binary
EXEC=studreg

all: $(OBJECTS)
    $(GCC) $(CCOPT) $(OBJECTS) -o $(EXEC)

%.o: %.cpp
    $(GCC) $(CCOPT) -c $< -o $@

clean:
    $(RM) *.o
    $(RM) studlist

-UU-:----F1 Makefile All L20 (GNUmakefil
```



# Makefile: Fullstendig

Liste av objektfiler vi skal bygge  
(hver for seg)

Vanlig variabel (her, binærfile)

«Pattern rule» for samtlige .cpp-filer  
som skal bli .o-filer

```
alfreb — developer@CppDevel: ~/CPP_Course_Privat...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

# Add more as you please
OBJECTS=student.o square.o studreg.o

# Name of the final binary
EXEC=studreg

all: $(OBJECTS)
    $(GCC) $(CCOPT) $(OBJECTS) -o $(EXEC)

%.o: %.cpp
    $(GCC) $(CCOPT) -c $< -o $@

clean:
    $(RM) *.o
    $(RM) studlist

-UU-:----F1  Makefile          All L20    (GNUmakefil
```



# Makefile: Fullstendig

Liste av objektfiler vi skal bygge  
(hver for seg)

Vanlig variabel (her, binærfile)

«Pattern rule» for samtlige .cpp-filer  
som skal bli .o-filer

```
alfreb — developer@CppDevel: ~/CPP_Course_Privat...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

# Add more as you please
OBJECTS=student.o square.o studreg.o

# Name of the final binary
EXEC=studreg

all: $(OBJECTS)
    $(GCC) $(CCOPT) $(OBJECTS) -o $(EXEC)

%.o: %.cpp
    $(GCC) $(CCOPT) -c $< -o $@

clean:
    $(RM) $(OBJECTS) $(EXEC)

- UU - : 7/20/2016 12:00:00 PM (GNUmakefile)
```

Filnavn på «input»-fila

Filnavn på «output» -fila



# Makefile: Fullstendig

Liste av objektfiler vi skal bygge  
(hver for seg)

Vanlig variabel (her, binærfile)

«Pattern rule» for samtlige .cpp-filer  
som skal bli .o-filer

Alt du trenger nå er å legge til flere  
filer under «OBJECTS».

Ny klasse «teacher.cpp»?  
Legg til «teacher.o»

```
alfreb — developer@CppDevel: ~/CPP_Course_Privat...
File Edit Options Buffers Tools Makefile Help
GCC=g++
CCOPT=-std=c++11

# Add more as you please
OBJECTS=student.o square.o studreg.o

# Name of the final binary
EXEC=studreg

all: $(OBJECTS)
    $(GCC) $(CCOPT) $(OBJECTS) -o $(EXEC)

%.o: %.cpp
    $(GCC) $(CCOPT) -c $< -o $@

clean:
    $(RM) $(OBJECTS) $(EXEC)

- UU - : (GNUmakefile
```

Filnavn på «input»-fila

Filnavn på «output» -fila



# Demo!

---

**student.cpp, studlist.cpp, Makefile**



# 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 allokert 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

I/O - Typisk sted det er naturligt med «try» - og naturligt å kaste exceptions.

```
studreg — emacs-24.3 — 56x25
File Edit Options Buffers Tools C++ Help

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()){
            // ...
            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;
    }
}

-UU-: **--F1 studlist.cpp 20% L33 (C++/1 Abbrev) --
```



# Demo på nett:

---

`exception_leak.cpp`