

Arguments of C++ Applications

g++ options, Libraries

`ifstream`

Using ROOT Libraries

Shahram Rahatlou



SAPIENZA
UNIVERSITÀ DI ROMA

<http://www.roma1.infn.it/people/rahatlou/programmazione++/>

Corso di Programmazione++

Roma, 27 April 2008

Options of g++

```
$ man g++
```

```
GCC(1) GNU GCC(1)
```

NAME

```
gcc - GNU project C and C++ compiler
```

SYNOPSIS

```
gcc [-c|-S|-E] [-std=standard]
    [-g] [-pg] [-Olevel]
    [-Wwarn...] [-pedantic]
    [-Idir...] [-Ldir...]
    [-Dmacro[=defn]...] [-Umacro]
    [-foption...] [-mmachine-option...]
    [-o outfile] infile...
```

Only the most useful options are listed here; see below for the remainder.
g++ accepts mostly the same options
as gcc.

DESCRIPTION

When you invoke GCC, it normally does preprocessing, compilation, assembly and linking. The ``overall options'' allow you to stop this process at an intermediate stage. For example, the `-c` option says not to run the linker. Then the output consists of object files output by the assembler.

Other options are passed on to one stage of processing. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; most of these are not documented here, since you rarely need to use any of them.

Some Already Familiar Options

```
$ ls -l color.*  
-rw-r--r-- 1 rahatlou None 601 May 22 13:10 color.cpp
```

- -E : stop after running pre-compiler to resolve pre-compiler directives
 - Don't compile nor link the binary

```
$ g++ -E -o color.pre-compiler color.cpp  
$ ls -l color.*  
-rw-r--r-- 1 rahatlou None 601 May 22 13:10 color.cpp  
-rw-r--r-- 1 rahatlou None 681002 May 23 11:19 color.pre-compiler
```

- -c : stop after compilation
 - Doesn't link so no executable is produced

```
$ g++ -c color.cpp  
$ ls -lrt color.*  
-rw-r--r-- 1 rahatlou None 601 May 22 13:10 color.cpp  
-rw-r--r-- 1 rahatlou None 681002 May 23 11:19 color.pre-compiler  
-rw-r--r-- 1 rahatlou None 29489 May 23 11:21 color.o
```

- -o : specify name of the output

```
$ g++ -c color.cpp  
$ ls -lrt  
-rw-r--r-- 1 rahatlou None 601 May 22 13:10 color.cpp  
-rw-r--r-- 1 rahatlou None 681002 May 23 11:19 color.pre-compiler  
-rw-r--r-- 1 rahatlou None 29489 May 23 11:21 color.o  
-rwxr-xr-x 1 rahatlou None 524423 May 23 11:22 a.out
```

Default name
of binary

Increasing Warning Level

```
// app1.cpp
#include <string>
#include <iostream>
int index() {
    int i = 27;
}

std::string name() {
    std::string str("test of g++ options");
    return str;

    // text after return
    int j = 56;
}

int main() {
    int i = index();
    std::string st = name();
    std::cout << "i: " << i
              << "st: " << st
              << std::endl;
    return 0;
}
```

Very often simple warnings are a clear signal there is something seriously wrong

```
$ g++ -o app1 app1.cpp
```

```
$ g++ -o app1 -Wall app1.cpp
```

```
app1.cpp: In function `int index()':
```

```
app1.cpp:6: warning: unused variable 'i'
```

```
app1.cpp:7: warning: control reaches end of non-void function
```

```
app1.cpp: In function `std::string name()':
```

```
app1.cpp:14: warning: unused variable 'j'
```

```
$ ./app1
```

```
i:0      st: test of g++ options
```

Value of index() is always 0! ignores completely you implementation!

Debug Symbols with -g

- Produce debugging information to be used by debuggers, e.g. GDB
 - larger binary
- Extremely useful when first developing your code
 - You can see the high level labels (your function names and variables)
- It slows down a bit the code but might pay off in development phase
- Once code fully tested you can remove this option and fully optimize

-g Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.

On most systems that use stabs format, -g enables use of extra debugging information that only GDB can use; this extra information makes debugging work better in GDB but will probably make other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use -gstabs+, -gstabs, -gxcoff+, -gxcoff, or -gvms (see below).

Unlike most other C compilers, GCC allows you to use -g with -O. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

Libraries of Compiled Code

- Libraries are simple archives that contain compiled code (object files)
- Two types of libraries
 - Static Library: used by linker to include compiled code in the executable at linking time.
 - Larger executable since includes ALL binary code run during execution
 - Does not require presence of libraries at runtime since all code already included in the executable
 - Shared Library: used by executable at runtime
 - The binary holds only references to functions in the libraries but the code is not included in the executable itself
 - Smaller executable size but REQUIRES library to be available at runtime
 - We will discuss shared libraries in a future lab session

Creating and Using Static Libraries

```
$ g++ -c Datum.cc
$ g++ -c Result.cc
$ g++ -c InputService.cc
$ g++ -c Calculator.cc

$ ar -r libMyLib.a Datum.o Result.o InputService.o Calculator.o
ar: creating libMyLib.a

$ ar tv libMyLib.a
rw-r--r-- 1003/513    1940 May 23 12:21 2006 Datum.o
rw-r--r-- 1003/513     748 May 23 12:21 2006 Result.o
rw-r--r-- 1003/513   4482 May 23 12:21 2006 InputService.o
rw-r--r-- 1003/513  22406 May 23 12:21 2006 Calculator.o

$ g++ -o wgtavg wgtavg.cpp -lMyLib -L.

$ ./wgtavg
```

Commonly Used g++ Options with External Libraries

- Usually when using external libraries you are provided with
 - path to directory where you can find include files
 - path to directory where you can find libraries
 - NO access to source code!
 - But you don't need the source code to compile. Only header files. Remember only interface matters!
- `-L` : path to directory containing libraries
 - `-L /usr/local/root/5.08.00/lib`
- `-I` : path to directory containing header files
 - `-I /usr/local/root/5.08.00/include`
- `-l` : specify name of libraries to be used at link time
 - `-l Core -lHbook`
 - you don't have to specify the prefix "lib" nor the extension ".a"

Optimizing Your Executable

- g++ offers many options to optimize your executable and reduce execution time
 - Compiler analyzes your code to determine the best execution path
 - Takes longer to compile with optimization
 - It's harder to debug an optimized program
 - Remember: your optimized and non-optimized executables MUST give the same results or you have a bug!

Options That Control Optimization

These options control various sorts of optimizations.

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

The compiler performs optimization based on the knowledge it has of the program. Using the `-funit-at-a-time` flag will allow the compiler to consider information gained from later functions in the file when compiling a function. Compiling multiple files at once to a single output file (and using `-funit-at-a-time`) will allow the compiler to use information gained from all of the files when compiling each of them.

Not all optimizations are controlled directly by a flag. Only optimizations that have a flag are listed.

Levels of Optimization

-O1 Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

With **-O**, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

-O turns on the following optimization flags: **-fdefer-pop -fmerge-constants -fthread-jumps -floop-optimize -fif-conversion -fif-conversion2 -fdelayed-branch -fguess-branch-probability -fcprop-registers**

-O also turns on **-fomit-frame-pointer** on machines where doing so does not interfere with debugging.

-O2 Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. The compiler does not perform loop unrolling or function inlining when you specify **-O2**. As compared to **-O**, this option increases both compilation time and the performance of the generated code.

-O2 turns on all optimization flags specified by **-O**. It also turns on the following optimization flags: **-fforce-mem -foptimize-sibling-calls -fstrength-reduce -fcse-follow-jumps -fcse-skip-blocks -frerun-cse-after-loop -frerun-loop-opt -fgcse -fgcse-lm -fgcse-sm -fgcse-las -fdelete-null-pointer-checks -fexpensive-optimizations -fregmove -fschedule-insns -fschedule-insns2 -fsched-interblock -fsched-spec -fcaller-saves -fpeephole2 -freorder-blocks -freorder-functions -fstrict-aliasing -funit-at-a-time -falign-functions -falign-jumps -falign-loops -falign-labels -fcrossjumping**

Please note the warning under **-fgcse** about invoking **-O2** on programs that use computed gotos.

-O3 Optimize yet more. **-O3** turns on all optimizations specified by **-O2** and also turns on the **-finline-functions**, **-fweb** and **-frename-registers** options.

-O0 Do not optimize. This is the default.

You should notice the difference
in your application when using **-O3**

Passing Arguments to C++ Applications

```
// app2.cpp

#include <iostream>
using namespace std;

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

    cout << "# of cmd line arguments argc: " << argc << endl;
    cout << "argv[0]: " << argv[0] << endl;

    cout << "Running " << argv[0] << endl;

    return 0;
}
```

```
$ g++ -o app2 app2.cpp
$ ./app2
# of cmd line arguments argc: 1
argv[0]: ./app2
Running ./app2
```

- **argc** is number of command line arguments
 - it includes the name of the application as well!
- **argv** is vector of pointers to characters
 - interprets each set of disjoint characters as a token

Passing non-string values

```
// args.cpp
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    cout << "# of cmd line arguments argc: " << argc << endl;
    cout << "argv[0]: " << argv[0] << endl;

    if(argc < 4 ) {
        cout << "Error... not enough arguments!" << endl;
        cout << "Usage: args <integer> <double> <string>" << endl;
        cout << "now exiting..." << endl;
        return -1; // can be used by user to determine error condition
    }

    int index = atoi( argv[1] );
    double mean = atof( argv[2] );
    std::string name( argv[3] );

    cout << "Running " << argv[0]
         << " with "
         << "index: " << index
         << ", mean: " << mean
         << ", name: " << name
         << endl;

    return 0;
}
```

atoi: converts char to int

atof: converts char to double

User responsibility to check
validity of arguments provided
at runtime

```
$ ./args
# of cmd line arguments argc: 1
argv[0]: ./args
Error... not enough arguments!
Usage: args <integer> <double> <string>
now exiting...
$ ./args 34
# of cmd line arguments argc: 2
argv[0]: ./args
Error... not enough arguments!
Usage: args <integer> <double> <string>
now exiting...
$ ./args 34 3
# of cmd line arguments argc: 3
argv[0]: ./args
Error... not enough arguments!
Usage: args <integer> <double> <string>
now exiting...
$ ./args 34 3.1322 sprogrammazione
# of cmd line arguments argc: 4
argv[0]: ./args
Running ./args with index: 34, mean: 3.1322, name: sprogrammazione
```

Input from file with `ifstream`

```
// readfile.cc

#include <iostream>
#include <fstream> // both input and output streams
#include <string>

using namespace std;

int main() {

    // file name
    const char filename[30] = "input.txt";

    // create object for input file
    ifstream infile(filename); //input file object

    // string to hold each line
    string line;

    // make sure input file is open otherwise exit
    if(!infile.is_open()) {
        cerr << "cant open input file" << endl;
        return -1;
    }
}
```

You need to create an output stream object which communicates with the file

Better than a plain file!
You can ask the object if the file is actually there

Parsing input lines with `sscanf`

```
// readfile.cc -- continued

// variables to read in from file at each line
char nome[30];
double val, errpos;
float errneg;

// loop over file until end-of-file
while(! infile.eof() ) {
    // get current line
    getline(infile,line);
    if( line == "\n" || line == "" ) continue;

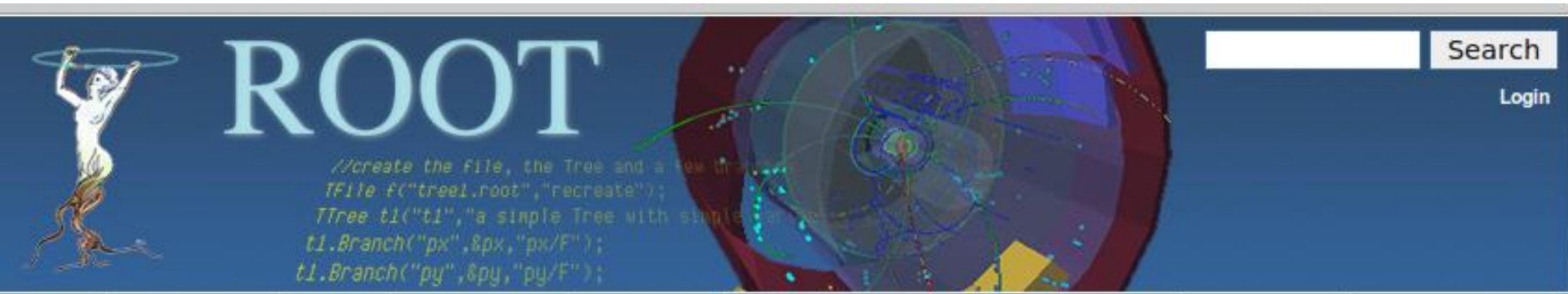
    // parse line with the provided format and put data in variables
    // NB: USING POINTERS TO VARIABLES
    // format: %s string    %f float    %lf double
    sscanf(line.c_str(),"%s %lf %lf %f",nome,&val,&errpos, &errneg);

    // print out for debug purposes
    cout << "nome: " << nome
         << "\tvalore: " << val << "\terr pos: " << errpos
         << "\terr neg: " << errneg << endl;
} // !eof

infile.close(); // close input file before exiting
return 0;
}
```

Boolean method tells you whether end-of-file has been reached yet

Using external Libraries: ROOT

The banner features the ROOT logo on the left, which is a stylized figure holding a ring. To the right of the logo is the word "ROOT" in large, glowing blue letters. Below the logo and text is a snippet of C++ code. On the far right, there is a search bar and a "Login" button. The background of the banner is a dark blue gradient with a 3D visualization of a particle detector or a complex tree structure on the right side.

ROOT

```
//create the file, the Tree and a few branches  
TFile f("tree1.root","recreate");  
TTree t1("t1","a simple Tree with simple branches");  
t1.Branch("px",&px,"px/F");  
t1.Branch("py",&py,"py/F");
```

[Search](#)
[Login](#)

How you can use ROOT

- ROOT provides extensive set of tools for data analysis
- Use 1D histograms to plot your data
- Use canvas provided by root to store the histogram as output in a file (eps or gif)
- Use root functionalities to make your plot nicer
 - Change color, labels, names, fonts
- Become familiar with using external libraries without access to source files

root: An object oriented data analysis framework



The image shows a screenshot of a web browser displaying the ROOT website. The browser's address bar shows the URL <http://root.cern.ch/drupal/>. The website header features the ROOT logo, a search bar, and a navigation menu with links for Home, What's New, About, Screenshots, Download, Documentation, Support, Forum, and Developers. Below the navigation menu, there are three main sections: Screenshots, Download, and Documentation. The Documentation section includes a red text overlay that reads "Reference guide is all you need!". At the bottom of the page, there is a "What's New" section with a sub-section for "Development release 5.23/04".

<http://root.cern.ch/drupal/>

Search

Login

```
//create the file, the Tree and a few branches
TFile f("tree1.root","recreate");
TTree t1("t1","a simple Tree with simple
t1.Branch("px",&px,"px/F");
t1.Branch("py",&py,"py/F");
```

Home | What's New | About | Screenshots | Download | Documentation | Support | Forum | Developers

Screenshots

Get a taste of ROOT's capabilities by sampling some [screenshots](#).

Download

Go ahead and [download](#) the latest build of ROOT.

Documentation

Get the inside scoop on how to fully utilize ROOT. Also, search the [Reference Guide](#), the [HowTo's](#) and the user forums.

What's New

Development release 5.23/04

development release

The development release of ROOT 5.23/04 is now available.

Reference guide is all you need!

Interface and Libraries are All You Need!

Reference Guide -> Class and Members Reference Guide -> Your Favorite Class



http://root.cern.ch/root/html/TH1F.html

CMS University Torrent Facebook Twitter YouTube

Bookmarks AutoLink AutoFill root TH1F

ROOT » HIST » HIST » TH1F

class TH1F: public TH1, public TArrayF

TH1F methods

TH1F : histograms with one float per channel. Maximum precision 7 digits

Function Members (Methods)

public:

```
TH1F ()
TH1F (const TVectorF& v)
TH1F (const TH1F& h1f)
TH1F (const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)
TH1F (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)
TH1F (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)
virtual ~TH1F ()
virtual void AddBinContent (Int_t bin)
virtual void AddBinContent (Int_t bin, Double_t w)
static TClass* Class ()
virtual void Copy (TObject& hnew) const
virtual TH1* DrawCopy (Option_t* option = "") const
virtual Double_t GetBinContent (Int_t bin) const
virtual Double_t GetBinContent (Int_t bin, Int_t) const
virtual Double_t GetBinContent (Int_t bin, Int_t, Int_t) const
virtual TClass* IsA () const
```

You can also the complete
html reference guide to your machine
for offline access (~300 MB)

Documentation During Our Lab Sessions

- From outside use root.cern.ch or <http://labcalc.phys.uniroma1.it/home/rahatlou/root/5.22.00/htmldoc/ClassIndex.html>
- From your working stations: <http://server/home/rahatlou/root/5.22.00/htmldoc/ClassIndex.html>

Using root libraries and header files

```
// app1.cpp

#include "TH1F.h"

int main() {

    TH1F h1;
    h1.Print();

    return 0;
}
```

```
$ setenv ROOTSYS /home/rahatlou/root/5.22.00
```

Needed at runtime to find libraries

```
$ setenv LD_LIBRARY_PATH $ROOTSYS/lib on tcsh
```

```
$ export LD_LIBRARY_PATH=$ROOTSYS/lib on bash
```

```
$ g++ -o app1 app1.cpp ` $ROOTSYS/bin/root-config --cflags --libs `
```

Provide path to header files and libraries

```
$ ./app1
```

```
TH1F.Print Name = , Entries= 0, Total sum= 0
```

root-config

```
$ $ROOTSYS/bin/root-config --cflags --libs  
-pthread -I/pool/home/rahatlou/root/5.11.02/include  
-L/pool/home/rahatlou/root/5.11.02/lib  
-lCore -lCint -lHist -lGraf -lGraf3d -lGpad -lTree -lRint  
-lPostscript -lMatrix -lPhysics -pthread -lm -ldl -rdynamic
```

- provides you with all options needed to compile and/or link your application
- Use at on command line with `` quotes instead of writing manually
- We will soon use makefiles to make such settings easier for users

Classes To Use in Your Application

- **TH1F**: 1D histogram
 - look at constructors
 - public methods to add data to histogram
 - public methods to add comments or change labels of axes
- **TCanvas**: canvas to draw your histogram
 - how to make one
 - changing properties such as color
 - drawing 1D histogram on a canvas
 - storing the canvas as a graphic file, e.g. eps or gif
- **TGraph**: dealing with asymmetric errors
 - more general than Histogram
 - Allows asymmetric errors but similar graphic functionalities as 1D histo

Simple Example with TH1

```
// app2.cpp
#include "TH1F.h"
#include "TCanvas.h"

int main() {

    // create histogram
    TH1F h1("h1","my first historgram",100,-6.0,6.0);

    // fill histogram with 10000 random gaussian numbers
    h1.FillRandom("gaus",10000);

    // add labels to axis
    h1.GetXaxis()->SetTitle("Gaussian variable");
    h1.GetYaxis()->SetTitle("arbitrary Units");

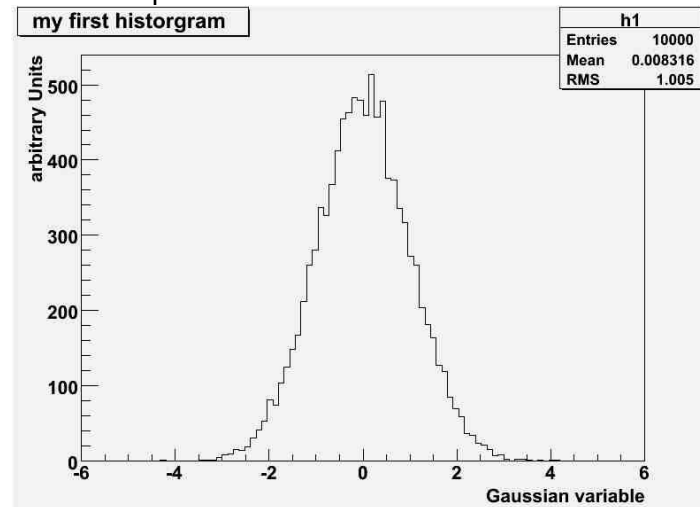
    // create a canvas to draw tour histogram
    TCanvas c1("c1","my canvas",800,600);

    // draw the histogram
    h1.Draw();

    // save canvas a JPG file
    c1.SaveAs("canvas.jpg");

    return 0;
}
```

```
$ g++ -Wall -o app2 app2.cpp `root-config --libs --cflags`
$ ./app2
```



A Few Tips about Using ROOT

- Look at the reference guide to find out what is provided by the interface
- Look at examples in <http://labcalc.phys.uniroma1.it/home/rahatlou/root/5.22.00/tutorials/>
- Start by simply creating new objects and testing them before making fancy use of many different classes

Common Mistakes in your Code

Where is the Mistake?

File Edit View Terminal Help

```
#ifndef IOService_h
#define IOService_h
```

```
#include <iostream>
#include "Datum.h"
#include <vector>
```

```
using std::vector;
```

```
class IOService {
public:
    IOService(){}
```

```
vector<Datum> readDataFromUser(){
```

```
    Datum inDatum;
```

```
    bool getvaluesFlag=true;
```

```
    double inValue;
```

```
    double inSigma;
```

```
    std::cout << "Insert the values and the errors."
```

```
        << " Value or sigma equal to 666 ends the input."
```

```
        << std::endl;
```

```
    while(getvaluesFlag==true){
```

```
        std::cout << "Value= ";
```

```
        std::cin >> inValue;
```

Never put `using` statements
in header files!

16,3

Top

Find the Mistake

File Edit View Terminal Help

```
#ifndef Datum_hh
#define Datum_hh
// Datum.hh
#include <iostream>
using namespace std;

class Datum {
public:
    Datum() {};
    Datum(double x, double y);
    // Datum(const Datum& datum);
    double value() { return value_; }
    double error() { return error_; }
    void display();
    // double significance();
private:
    double value_;
    double error_;
};

#endif
```

Never put `using` statements
in header files!

"Datum.hh" [readonly] 21L, 359C

1,1

All

File Edit View Terminal Help

```
#ifndef Dati_h
#define Dati_h
//Dati.h
#include <iostream>
#include <vector>

using namespace std;

class Dati {
    double value_, signal_, sigmar_;
public:

    Dati();
    Dati(double val, double signal, double sigmar);

    void setdati(double val, double signal, double sigmar);

    double value();
    double signal();
    double sigmar();

    double significancel();
    double significancer();
};
#endif
~
"Dati.h" [readonly] 25L, 394C
```

Never put `using` statements
in header files!

Two methods with almost
exactly the same name

How can I understand what they do??