

Class Destructors

constant member functions

C++ Application and command line options

External libraries: first examples with ROOT

Shahram Rahatlou

Computing Methods in Physics

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

Anno Accademico 2019/20



SAPIENZA
UNIVERSITÀ DI ROMA

Who and When Calls the Destructor?

Constructors are called by compiler when new objects are created

```
// app1.cpp
#include "Counter.h"
#include <string>

int main() {

    Counter c1( std::string("c1") );
    Counter c2( std::string("c2") );
    Counter c3( std::string("c3") );

    c2.increment(135);
    c1.increment(5677);

    c1.print();
    c2.print();
    c3.print();

    return 0;
}
```

Create in order objects c1, c2, and c3

Destructors are called implicitly by compiler when objects go out of scope!

Destructors are called in reverse order of creation

```
$ g++ -c Counter.cc
$ g++ -o app1 app1.cpp Counter.o
$ ./app1
Counter::Counter() called for Counter c1
Counter::Counter() called for Counter c2
Counter::Counter() called for Counter c3
Counter::print(): name: c1 value: 5677
Counter::print(): name: c2 value: 135
Counter::print(): name: c3 value: 0
Counter::~Counter() called for Counter c3
Counter::~Counter() called for Counter c2
Counter::~Counter() called for Counter c1
```

Destruct c3, c2, and c1

Another Example of Destructors

```
// app2.cpp
#include "Counter.h"
#include <string>

int main() {

    Counter c1( std::string("c1") );

    int count = 344;

    if( 1.1 <= 2.02 ) {
        Counter c2( std::string("c2") );

        Counter c3( std::string("c3") );
        if( count == 344 ) {
            Counter c4( std::string("c4") );
        }

        Counter c5( std::string("c5") );

        for(int i=0; i<3; ++i) {
            Counter c6( std::string("c6") );
        }
    }

    return 0;
}
```

```
$ g++ -o app2 app2.cpp Counter.o
$ ./app2
Counter::Counter() called for Counter c1
Counter::Counter() called for Counter c2
Counter::Counter() called for Counter c3
Counter::Counter() called for Counter c4
Counter::~Counter() called for Counter c4
Counter::Counter() called for Counter c5
Counter::Counter() called for Counter c6
Counter::~Counter() called for Counter c6
Counter::Counter() called for Counter c6
Counter::~Counter() called for Counter c6
Counter::Counter() called for Counter c6
Counter::~Counter() called for Counter c6
Counter::Counter() called for Counter c5
Counter::~Counter() called for Counter c3
Counter::~Counter() called for Counter c2
Counter::~Counter() called for Counter c1
```

Using new and delete Operators

```
// app6.cpp
#include "Counter.h"
#include "Datum.h"
#include <iostream>
using namespace std;

int main() {

    Counter c1("c1");

    Counter* c2 = new Counter("c2");
    c2->increment(6);

    Counter* c3 = new Counter("c3");

    Datum d1(-0.3,0.07);

    Datum* d2 = new Datum( d1 );
    d2->print();

    delete c2; // de-allocate memory!
    delete c3; // de-allocate memory!
    delete d2;

    return 0;
}
```

```
$ g++ -o app6 app6.cpp Datum.o Counter.o
$ ./app6
Counter::Counter() called for Counter c1
Counter::Counter() called for Counter c2
Counter::Counter() called for Counter c3
datum: -0.3 +/- 0.07
Counter::~Counter() called for Counter c2
Counter::~Counter() called for Counter c3
Counter::~Counter() called for Counter c1
```

Order of calls to destructors has changed!

delete calls explicitly the destructor of the object to de-allocate memory

Vital for objects holding pointers to dynamically allocated memory

Why no message when destructing d2 ?

constant Member Functions

- Enforce principle of least privilege
 - Give privilege ONLY if needed
- **const** member functions cannot
 - modify data members
 - cannot be called on non-constant objects
- **const** member functions tell user, the function only ‘uses’ the input data or data members but makes **no changes to data members**
- Pay attention which function can be called on which objects
 - Objects can be constant
 - You can not modify a constant object
 - calling non-constant methods on constant objects does not make sense!

Datum Class and const Member Functions

```
class Datum {  
public:  
    Datum();  
    Datum(double x, double y);  
    Datum(const Datum& datum);  
  
    double value() { return value_; }  
    double error() { return error_; }  
    double significance();  
    void print();  
  
    void setValue(double x) { value_ = x; }  
    void setError(double x) { error_ = x; }  
  
private:  
    double value_;  
    double error_;  
};
```

Which methods
could become constant?

Datum Class with const Methods

All methods that only return a value and do not change the attributes of an object!

All getters can be constant

```
#ifndef Datum1_h
#define Datum1_h
// Datum1.h
#include <iostream>
using namespace std;

class Datum {
public:
    Datum();
    Datum(double x, double y);
    Datum(const Datum& datum);

    double value() const { return value_; }
    double error() const { return error_; }
    double significance() const;
    void print() const;

    void setValue(double x) { value_ = x; }
    void setError(double x) { error_ = x; }

private:
    double value_;
    double error_;
};

#endif
```

what about setter
methods?

```
#include "Datum1.h"
#include <iostream>

Datum::Datum() {
    value_ = 0.; error_ = 0.;
}

Datum::Datum(double x, double y) {
    value_ = x; error_ = y;
}

Datum::Datum(const Datum& datum) {
    value_ = datum.value_;
    error_ = datum.error_;
}

double
Datum::significance() const {
    return value_/error_;
}

void Datum::print() const {
    using namespace std;
    cout << "datum: " << value_
        << " +/- " << error_ << endl;
}
```

Typical error with constant methods

```
#ifndef Datum2_h
#define Datum2_h
// Datum2.h
#include <iostream>
using namespace std;

class Datum {
public:
    Datum();
    Datum(double x, double y);
    Datum(const Datum& datum);

    double value() const { return value_; }
    double error() const { return error_; }
    double significance() const;
    void print() const;

    void setValue(double x) const { value_ = x; }
    void setError(double x) const { error_ = x; }

private:
    double value_;
    double error_;
};

#endif
```

setters can never be constant!

Setter method is used to modify data members

Similarly constructors and destructors can not be constant

```
$ g++ -c Datum2.cc
In file included from Datum2.cc:1:
Datum2.h: In member function `void Datum::setValue(double) const':
Datum2.h:18: error: assignment of data-member `Datum::value_' in read-only structure
Datum2.h: In member function `void Datum::setError(double) const':
Datum2.h:19: error: assignment of data-member `Datum::error_' in read-only structure
```

Example of Error using non-constant functions

```
#ifndef Datum4_h
#define Datum4_h
// Datum4.h
#include <iostream>
#include <string>
using namespace std;

class Datum {
public:
    Datum();
    Datum(double x, double y);
    Datum(const Datum& datum);

    double value() const { return value_; }
    double error() const { return error_; }
    double significance() const;

    void print(const std::string& comment) ;
    void setValue(double x) { value_ = x; }
    void setError(double x) { error_ = x; }

private:
    double value_;
    double error_;
};

#endif
```

print MUST have been constant!
bad design of the class!

```
void Datum::print(const std::string& comment) {
    using namespace std;
    cout << comment << ": " << value_
        << " +/- " << error_ << endl;
```

```
// appl.cpp

#include "Datum4.h"

int main() {

    Datum d1(-67.03, 32.12 );
    const Datum d2(-67.03, 32.12 );

    d1.print("datum");

    d2.print("const datum");

    return 0;
}
```

```
$ g++ -o appl appl.cpp Datum4.o
appl.cpp: In function `int main()':
appl.cpp:12: error: passing `const Datum' as `this'
argument of `void Datum::print(const std::string&)'
discards qualifiers
```

Default Values for Methods

- Functions (not only member functions in classes) might be often invoked with recurrent values for their arguments
- It is possible to provide default values for arguments of any function in C++
 - Default arguments must be provided the first time the name of the function occurs
 - In declaration if separate implementation
 - In definition if the function is declared and defined at the same time
- Only the right-most argument can be omitted
 - Including all arguments to the right of omitted argument

Example of Default Values

```
// Counter.h

class Counter {
public:
    Counter();
    int value();
    void reset();
    void increment();
    void increment(int step);

private:
    int count_;
};
```

Two increment() methods
but very similar functionality

increment() is a special case of
increment(int step) with step=1

Why two different methods?

```
// Counter.cc
// include class header files
#include "Counter.h"

// include any additional header files
// needed in the class
// definition
#include <iostream>
using std::cout;
using std::endl;

Counter::Counter() {
    count_ = 0;
}

int Counter::value()  {
    return count_;
}

void Counter::reset() {
    count_ = 0;
}

void Counter::increment()  {
    count_++;
}

void Counter::increment(int step) {
    count_ = count_+step;
}
```

Default Value for Counter::increment(int step)

```
#ifndef Counter_Old_h_
#define Counter_Old_h_
// CounterOld.h

class Counter {
public:
    Counter();
    int value();
    void reset();
    void increment(int step = 1);
private:
    int count_;
};

#endif
```

Bad Practice!
Name of class
different from name
of file

```
// CounterOld.cc
#include "CounterOld.h"
#include <iostream>
using std::cout;
using std::endl;

Counter::Counter() {
    count_ = 0;
}

int Counter::value() {
    return count_;
}

void Counter::reset() {
    count_ = 0;
}

void Counter::increment(int step) {
    count_ = count_+step;
}
```

```
// app3.cpp
#include "CounterOld.h" // old counter class
#include <iostream>
using namespace std;

int main() {

    Counter c1;

    c1.increment(); // no argument
    cout << "counter: " << c1.value() << endl;

    c1.increment(14); // provide argument, same function
    cout << "counter: " << c1.value() << endl;

    return 0;
}
```

```
$ g++ -c CounterOld.cc
$ g++ -o app3 app3.cpp
CounterOld.o
.$ ./app3
counter: 1
counter: 15
```

Ambiguous Use of Default Arguments

```
#ifndef Datum_h
#define Datum_h
// Datum.h
#include <iostream>
using namespace std;

class Datum {
public:
    //Datum();
    Datum(double x=1.0, double
y=0.0);
    Datum(const Datum& datum);
    double value() { return value_; }
    double error() { return error_; }
    double significance();

private:
    double value_;
    double error_;
};

#endif
```

Does it make sense to have default value and error?

```
$ g++ -c Datum.cc
$ g++ -o app4 app4.cpp Datum.o
$ ./app4
datum: -0.23 +/- 0.05
datum: 5.23 +/- 0
datum: 1 +/- 0
```

```
#include "Datum.h"

Datum::Datum(double x, double y) {
    value_ = x;
    error_ = y;
}

Datum::Datum(const Datum& datum) {
    value_ = datum.value_;
    error_ = datum.error_;
}

double
Datum::significance() {
    return value_/error_;
```

```
#include "Datum.h"
int main() {

    Datum d1(-0.23, 0.05); // provide arguments
    d1.print();

    Datum d2(5.23); // default error ...
    d2.print();

    Datum d3; // default value and error!
    d3.print();
    return 0;
}
```

Don't Abuse Default Arguments!

- Default values must be used for functions very similar in functionality and with obvious default values
- If default values are not intuitive for user think twice before using them!
- Quite often different constructors correspond to DIFFERENT ways to create an object
 - Default values could be misleading
- If arguments are physical quantities ask yourself: is the default value meaningful and useful for everyone?

*C++ Applications
command line arguments,
g++ options, I/O to/from file
External Libraries: ROOT*

Options of g++

SYNOPSIS

```
gcc [ | | ] [ standard ]
      [ ] [ ] [ level ]
      [ warn... ] [ ]
      [ dir... ] [ dir... ]
      [ macro[=defn]... ] [ macro ]
      [ option... ] [ machine-option... ]
      [ outfile ] [ @file ] infile...
```

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

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.

Most of the command line options that you can use with GCC are useful for C programs; when an option is only useful with another language (usually C++), the explanation says so explicitly. If the description for a particular option does not mention a source language, you can use that option with all supported languages.

The program accepts options and file names as operands. Many options have multi-letter names; therefore multiple single-letter options may not be grouped: is very different from `-d -c`.

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 your 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 might discuss shared libraries in a future lecture

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 -fre-run-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

```
// app12.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>
#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 inut file" << endl;
        return -1;
    }
}
```

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;
}
```

ROOT for data analysis

ROOT: An object oriented data analysis framework



ROOT
Data Analysis Framework

<http://root.cern.ch>

Google Custom Search

[Download](#) [Documentation](#) [News](#) [Support](#) [About](#) [Development](#) [Contribute](#)



[Getting Started](#)



[Reference Guide](#)



[Forum](#)



[Gallery](#)

Reference guide is all you need!

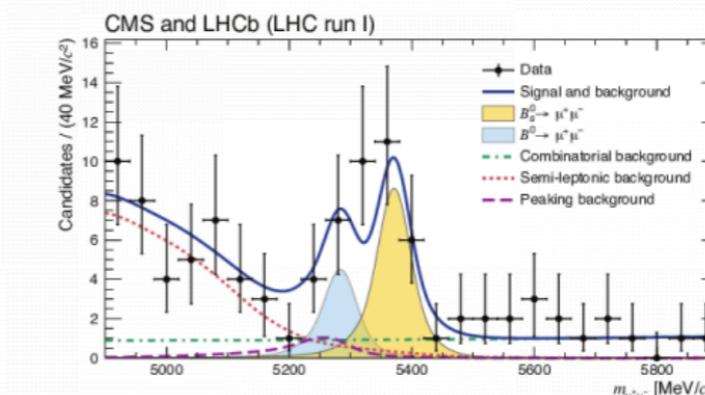
ROOT is ...

A modular scientific software toolkit. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

[Start from examples](#) or [try it in your browser!](#)

 [Download](#)

or [Read More ...](#)



[Previous](#) [Resume](#) [Next](#)

Under the Spotlight

2018-01-17 [ROOT Users' Workshop 2018](#)

The ROOT team would like to invite you to the [11th ROOT Users' Workshop](#).

2017-08-03 [The ROOT Docker container \(alpha version\)](#)

Do you like [Docker](#)? Would you like to use ROOT? We provide an *alpha* version of the ROOT [Docker](#) container!

2016-09-05 [Get the most out of the ROOT tutorials!](#)

Other News

2018-07-11 [RDataFrame session at CHEP 2018](#)

2018-06-15 [ROOT::RWhy!](#)

2017-03-08 [Development release 6.09/02 is out!](#)

2016-04-16 [The status of reflection in C++](#)

Latest Releases

Release 6.14/04 - 2018-08-23

ROOT Reference Guide



ROOT 6.15/01
Reference Guide

<https://root.cern/doc/master/>

ROOT Home Main Page Tutorials Functional Parts Namespaces ▾ All Classes ▾ Files ▾ Release Notes TH

ROOT Reference Documentation

Introduction

Welcome to ROOT

This documentation describes the software classes and functions that make up the ROOT software system as well as an introduction of ROOT, for this please refer to the [ROOT User Guides and Manuals](#). This documentation is generated and kept up to date. The version of ROOT corresponding to this documentation is indicated at the page heading. You may also find in [reference documentation page](#) pointers to reference manuals for other ROOT versions.

How to use this reference documentation

The [User's Classes](#) in the top bar provides the user API, mainly the list of main Users' classes organized by module or functionality. The full list of classes, both for the public API and for the implementation details are available under the [All Classes](#) tab. A classification of classes based on their C++ namespace can be found under the [Namespace](#) tab. The fully indexed list of all source code is available under the tab [Files](#).

ROOT provides other types of documentation:

- A general [Users Guide](#) is provided for a more in depth explanation of concepts and functionality available in the ROOT system.
- A number of topical [User Guides and Manuals](#) for various components of the system.
- A rich set of ROOT [tutorials](#) and [code examples](#) are offered to developers to exercise specific functionality.
- A rich set of [HowTo's](#) is also present to discuss issues commonly faced by ROOT users.

Caveat

We have moved recently to generate the documentation with Doxygen. To achieve this the comments in the source code needed to be formatted and written specifically for Doxygen to generate proper documentation. If you find missing documentation or inaccuracies please report them to our [bug tracker](#). Detailed instructions on how to submit a bug can be found [here](#).

The ROOT Mathematical Libraries
The Dataset Stager
th ROOT::R::TRInterface
TH1
TH1.cxx
TH1.h
TH1C
TH1C.h
TH1D
TH1D.h
TH1DModel
TH1Editor
TH1Editor.cxx
TH1Editor.h



1D Histogram of floats: TH1F

TH1F Class Reference

Histogram Library

[LIST OF ALL MEMBERS](#) | [PUBLIC MEMBER FUNCTIONS](#) | [PROTECTED MEMBER FUNCTIONS](#) | [FRIENDS](#) | [LIST OF ALL FRIENDS](#)

1-D histogram with a float per channel (see [TH1](#) documentation)}

Definition at line [571](#) of file [TH1.h](#).

Public Member Functions

[TH1F \(\)](#)

Constructor. [More...](#)

[TH1F \(const char *name, const char *title, Int_t nbinsx, Double_t xlow, Double_t xup\)](#)

Create a 1-Dim histogram with fix bins of type float (see [TH1::TH1](#) for explanation of parameters) [More...](#)

[TH1F \(const char *name, const char *title, Int_t nbinsx, const Float_t *xbins\)](#)

Create a 1-Dim histogram with variable bins of type float (see [TH1::TH1](#) for explanation of parameters) [More...](#)

[TH1F \(const char *name, const char *title, Int_t nbinsx, const Double_t *xbins\)](#)

Create a 1-Dim histogram with variable bins of type float (see [TH1::TH1](#) for explanation of parameters) [More...](#)

[TH1F \(const TVectorF &v\)](#)

Create a histogram from a TVectorF by default the histogram name is "TVectorF" and title = "". [More...](#)

[TH1F \(const TH1F &h1f\)](#)

Copy Constructor. [More...](#)

[virtual ~TH1F \(\)](#)

Destructor. [More...](#)

[virtual void AddBinContent \(Int_t bin\)](#)

Increment bin content by 1. [More...](#)

[virtual void AddBinContent \(Int_t bin, Double_t w\)](#)

Increment bin content by a weight w. [More...](#)

[virtual void Copy \(TObject &hnew\) const](#)

Copy this to newth1. [More...](#)

Using ROOT

- ▷ Use classes from root in your application
- ▷ 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

Interface and Libraries are All You Need!

Public Member Functions

[TH1F \(\)](#)

Constructor. More...

<https://root.cern/doc/master/classTH1F.html>

[TH1F \(const char *name, const char *title, Int_t nbinsx, Double_t xlow, Double_t xup\)](#)

Create a 1-Dim histogram with fix bins of type float (see [TH1::TH1](#) for explanation of parameters) More...

[TH1F \(const char *name, const char *title, Int_t nbinsx, const Float_t *xbins\)](#)

Create a 1-Dim histogram with variable bins of type float (see [TH1::TH1](#) for explanation of parameters) More...

[TH1F \(const char *name, const char *title, Int_t nbinsx, const Double_t *xbins\)](#)

Create a 1-Dim histogram with variable bins of type float (see [TH1::TH1](#) for explanation of parameters) More...

[TH1F \(const TVectorF &v\)](#)

Create a histogram from a TVectorF by default the histogram name is "TVectorF" and title = "". More...

[TH1F \(const TH1F &h1f\)](#)

Copy Constructor. More...

virtual [~TH1F \(\)](#)

Destructor. More...

virtual void [AddBinContent \(Int_t bin\)](#)

Increment bin content by 1. More...

virtual void [AddBinContent \(Int_t bin, Double_t w\)](#)

Increment bin content by a weight w. More...

virtual void [Copy \(TObject &hnew\) const](#)

Copy this to newth1. More...

TH1F & [operator= \(const TH1F &h1\)](#)

Operator =. More...

virtual void [Reset \(Option_t *option=""\)](#)

Reset. More...

virtual void [SetBinsLength \(Int_t n=-1\)](#)

Set total number of bins including under/overflow Reallocate bin contents array. More...

◆ [TH1F\(\) \[2/6\]](#)

```
TH1F::TH1F ( const char * name,  
              const char * title,  
              Int_t      nbinsx,  
              Double_t   xlow,  
              Double_t   xup  
            )
```

Create a 1-Dim histogram with fix bins of type float (see [TH1::TH1](#) for explanation of parameters)

Definition at line 9349 of file [TH1.cxx](#).

Installing ROOT

- ▷ Information for downloading ROOT available at
<https://root.cern/downloading-root>
- ▷ Binaries provided for variety of architecture and OS
 - See latest release 6.18/04: <https://root.cern/content/release-61804>

Binary distributions

Platform	Files	Size
CentOS Cern 7 gcc4.8	root_v6.18.04.Linux-centos7-x86_64-gcc4.8.tar.gz	137M
Linux fedora29 gcc8.3	root_v6.18.04.Linux-fedora29-x86_64-gcc8.3.tar.gz	155M
Linux fedora30 gcc9.2	root_v6.18.04.Linux-fedora30-x86_64-gcc9.2.tar.gz	158M
Ubuntu 14 gcc4.8	root_v6.18.04.Linux-ubuntu14-x86_64-gcc4.8.tar.gz	137M
Ubuntu 16 gcc5.4	root_v6.18.04.Linux-ubuntu16-x86_64-gcc5.4.tar.gz	144M
Ubuntu 18 gcc7.4	root_v6.18.04.Linux-ubuntu18-x86_64-gcc7.4.tar.gz	154M
OsX 10.13 clang100	root_v6.18.04.macosx64-10.13-clang100.dmg	130M
OsX 10.13 clang100	root_v6.18.04.macosx64-10.13-clang100.tar.gz	129M
OsX 10.14 clang100	root_v6.18.04.macosx64-10.14-clang100.dmg	131M
OsX 10.14 clang100	root_v6.18.04.macosx64-10.14-clang100.tar.gz	130M
Windows Visual Studio 2019 (dbg)	root_v6.18.04.win32.vc16.debug.exe	181M
Windows Visual Studio 2019 (dbg)	root_v6.18.04.win32.vc16.debug.zip	275M
Windows Visual Studio 2019	root_v6.18.04.win32.vc16.exe	79M
Windows Visual Studio 2019	root_v6.18.04.win32.vc16.zip	106M

Windows

Windows 7/Vista/XP/NT/2000 are supported. We offer two packaging types:

- **exe**: a regular Windows installer package also setting up the required environment variables. With uninstall via "Control Panel" / "Add or Remove Programs". Simply download and start, or open directly. You can double-click ROOT to start it, ROOT files get registered with Windows.
- **tar**: the traditional variant. Unpack e.g. with [7zip](#). Start ROOT in a Microsoft Visual Studio Prompt (in Start / Programs / Microsoft Visual Studio / Tools). If you installed ROOT to C:\root then call C:\root\bin\thisroot.bat before using ROOT to set up required environment variables.

Important installation notes

- You must download the binary built with the exact same version of Visual Studio than the one installed on your system.
- Do not untar in a directory with a name containing blank characters.
- Take the release version if performance matters.
- If you want to debug your code you need the ROOT debug build (you cannot mix release / debug builds due to a Microsoft restriction).

Using root libraries and header files

```
// app13.cc  
  
#include "TH1F.h"  
  
int main() {  
  
    TH1F h1;  
    h1.Print();  
  
    return 0;  
}
```

```
csh> setenv ROOTSYS /home/rahatlou/root/5.11.02
```

Needed at runtime to find libraries

```
bash> export ROOTSYS=/home/rahatlou/root/5.11.02
```

```
csh> setenv LD_LIBRARY_PATH $ROOTSYS/lib
```

Provide path to header files and libraries

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

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

```
$ ./app1
```

```
TH1.Print Name = , Entries= 0, Total sum= 0
```

root-config

```
> $ROOTSYS/bin/root-config --cflags --libs  
-pthread -stdlib=libc++ -std=c++11 -m64 -I/Users/  
rahatlou/Library/root/v6.14.00/include -L/Users/rahatlou/  
Library/root/v6.14.00/lib -lCore -lImt -lRIO -lNet -lHist  
-lGraf -lGraf3d -lGpad -lROOTDataFrame -lROOTVecOps -  
lTree -lTreePlayer -lRint -lPostscript -lMatrix -lPhysics  
-lMathCore -lThread -lMultiProc -lpthread -stdlib=libc++  
-lm -ldl
```

- ▷ 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

First classes to use

- ▷ **TH1F: 1D histogram**
 - look at constructors
 - public methods to add data to histogram
 - public methods to add comments or change labels of axes
- ▷ **TCavnvas: 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

Simple Example with TH1

```
// app14.cc
#include "TH1F.h"
#include "TCanvas.h"

int main() {

    // create histogram
    TH1F h1("h1","my first histogram",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",1024,800);

    // draw the histogram
    h1.Draw();

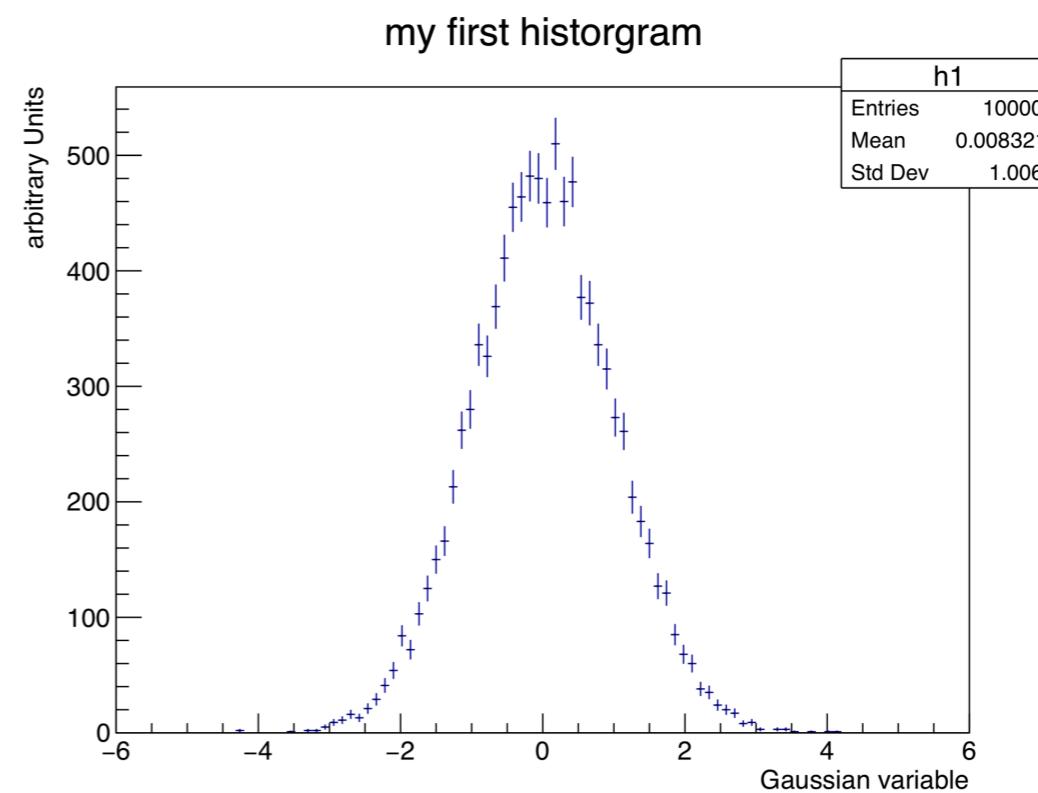
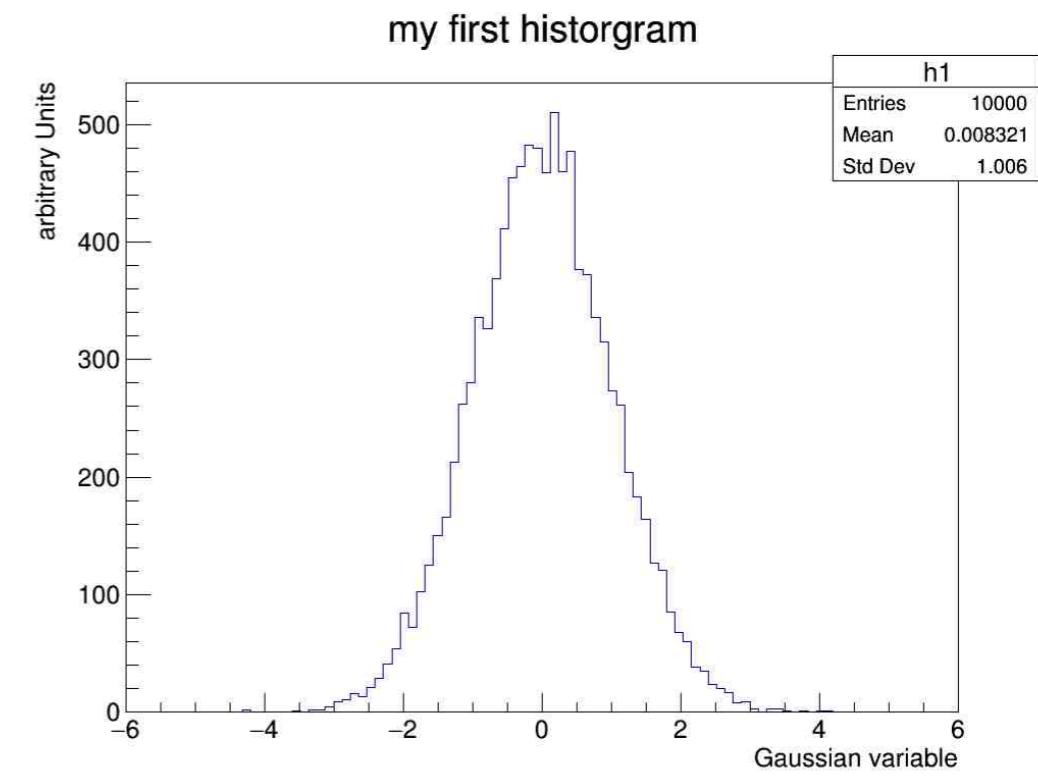
    // save canvas a JPG file
    c1.SaveAs("canvas.jpg");

    // change fill color to blue
    h1.SetFillColor(kBlue);

    // draw again the histogram
    h1.Draw();
    c1.SaveAs("canvas-blue.jpg");

    // dra histograms as points with errors
    h1.Draw("pe");

    // save canvas a PDF file
    c1.SaveAs("canvas-points.pdf");
    return 0;
}
```



```
$ g++ -Wall -o app4 app4.cc `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
- ▷ Start by simply creating new objects and testing them before making fancy use of many different classes