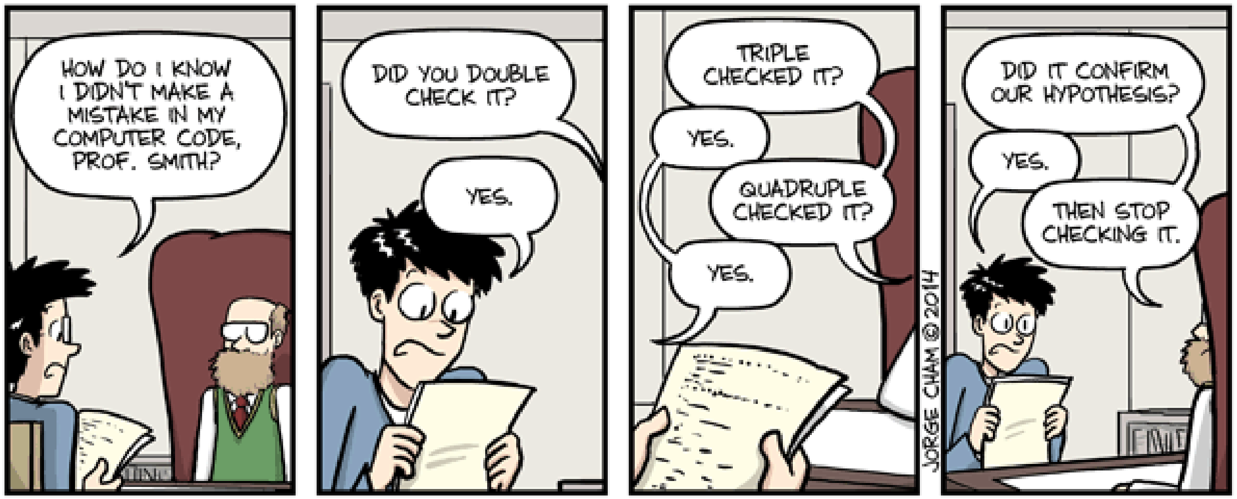
Outline 2017/03/24

- How to avoid crashing
- How to report bugs
- How to debug a program
  - l/s trace
  - ldd
  - nm
  - valgrind
  - ddd
- Make and CMake



Avoid crashing (whenever possible)

A program that crashes badly:

1) could create “core” dump filling the disk;
2) could create a problems when part of a procedure;
3) force the developer to search for the bug with few clues.

Hence: try to catch possible errors!

Avoid crashing (whenever possible)

For example:

A program needs to connect to a given server to retrieve a certain number.

The program crashes because the number used in the code has no sense...

- is it a problem on the server (the stored number was wrong)?
- is it a problem with the connection (somebody unplugged the network cable)?
- is it a problem with the server filename (user provided a mispelled name)?
- is it a bug in the code (overriding our variable)?

Avoid crashing (whenever possible)

A program that crashes is like a person who dies while eating mushrooms... were the mushroom venenous? did the person eat too much? was it uncorrelated to the mushrooms?

A program that exit with an error is like a person who feels sick but can still talk and tell you what is wrong. The person can tell you that he/she spit the mushrooms since they had a bitter flavour; the person can tell you that he/she was eating mushrooms when a lamp felt on his/her head, etc.

Avoid crashing (whenever possible)

How to avoid a crash:

- is it a problem on the server (the stored number was wrong)?

  check the result of the query, is it a number? if you expect the number to be in a certain range, is it in the range? if not, print out an error line stating what is wrong and exit.

- is it a problem with the connection (somebody unplugged the network cable)?
- is it a problem with the server filename (user provided a mispelled name)?

  check the connection! when you try to connect to the server what it the answer? no connection available? no such a server? print out an error line and exit.

Bugs reporting

http://wwwusers.ts.infn.it/~mocchiut/bugs/bugs.html

- It doesn’t work
- Show me
- Show me how to show myself
- Works for me, what goes wrong?
- So then I tried...
- I think the tachyon modulation must be wrongly polarized
- That’s funny, I did it one a moment ago!
- So I loaded the disk on to my Windows...

Bugs reporting

- **It doesn't work**

  Give the programmer some credit for basic intelligence: if the program really didn't work at all, they would probably have noticed. Since they haven't noticed, it must be working for them. Therefore, either you are doing something differently from them, or your environment is different from theirs. They need information; providing this information is the purpose of a bug report. More information is almost always better than less.

Bugs reporting

- **Show me**

  One of the very best ways you can report a bug is by showing it to the programmer. Stand them in front of your computer, fire up their software, and demonstrate the thing that goes wrong. Let them watch you start the machine, watch you run the software, watch how you interact with the software, and watch what the software does in response to your inputs

Bugs reporting

- **Show me how to show myself**

  If you have to report a bug to a programmer who can't be present in person, the aim of the exercise is to enable them to *reproduce* the problem. You want the programmer to run their own copy of the program, do the same things to it, and make it fail in the same way.

  So tell them exactly what you did. If it's a graphical program, tell them which buttons you pressed and what order you pressed them in. If it's a program you run by typing a command, show them precisely what command you typed.

Bugs reporting

- **Works for me, what goes wrong?**

  Possibly the fault doesn't show up on every computer; your system and theirs may differ in some way. Possibly you have misunderstood what the program is supposed to do, and you are both looking at exactly the same display but you think it's wrong and they know it's right.

  Describe what happened. Tell them exactly what you saw. Tell them why you think what you saw is wrong; better still, tell them exactly what you *expected* to see. If you say "and then it went wrong", you have left out some very important information.

Bugs reporting

- **So then I tried...**

  Some users are like a mongoose (mangusta) backed into a corner: with its back to the wall and seeing certain death staring it in the face, it attacks frantically, because doing something has to be better than doing nothing. This is not well adapted to the type of problems computers produce.

  Instead of being a mongoose, be an antelope. When an antelope is confronted with something unexpected or frightening, it freezes. It stays absolutely still and tries not to attract any attention, while it stops and thinks and works out the best thing to do (if antelopes had a technical support line, it would be telephoning it at this point). Then, once it has decided what the safest thing to do is, it does it.

Bugs reporting

- **I think the tachyon modulation must be wrongly polarized**

  “Doctor, I need a prescription for Hydroyoyodyne”. People know not to say that to a doctor: you describe the symptoms, the actual discomforts and aches and pains and rashes and fevers, and you let the doctor do the diagnosis of what the problem is and what to do about it.

  It's the same with programmers. Providing your own diagnosis might be helpful sometimes, but *always state the symptoms*. The diagnosis is an optional extra, and not an alternative to giving the symptoms.

Bugs reporting

- **That's funny, I did it one a moment ago!**

  Most intermittent faults are not truly intermittent. Most of them have some logic somewhere. Some might occur when the machine is running out of memory, some might occur when another program tries to modify a critical file at the wrong moment, and some might occur only in the first half of every hour!

  Try to remember as much detail as you can about what you were doing to it when it did fall over, and if you see any patterns, mention them. Anything you can provide has to be some help.

Bugs reporting

- **So I loaded the disk on to my Windows...**

  Writing clearly is essential in a bug report. If the programmer can't tell what you meant, you might as well not have said anything:

  – *Be specific*. If you can do the same thing two different ways, state which one you used. "I selected Load" might mean "I clicked on Load" or "I pressed Alt-L". Say which you did. Sometimes it matters.

  – *Be verbose*. Give more information rather than less. If you say too much, the programmer can ignore some of it. If you say too little, they have to come back and ask more questions. One bug report I received was a single sentence; every time I asked for more information, the reporter would reply with another single sentence. It took me several weeks to get a useful amount of information, because it turned up one short sentence at a time.

Bugs reporting

- **So I loaded the disk on to my Windows...**
  - *Be careful of pronouns*. Don't use words like "it", or references like "the window", when it's unclear what they mean. Consider this: "I started FooApp. It put up a warning window. I tried to close it and it crashed." It isn't clear what the user tried to close. Did they try to close the warning window, or the whole of FooApp? It makes a difference. Instead, you could say "I started FooApp, which put up a warning window. I tried to close the warning window, and FooApp crashed." This is longer and more repetitive, but also clearer and less easy to misunderstand.
  - *Read what you wrote*. Read the report back to yourself, and see if *you* think it's clear. If you have listed a sequence of actions which should produce the failure, try following them yourself, to see if you missed a step.

Debugging

First and more important rule of all:

READ CAREFULLY <u>WARNINGS</u> AND <u>ERRORS</u>!

Debugging

First and more important rule of all:

READ CAREFULLY

<u>WARNINGS</u> AND <u>ERRORS!</u>

```
g++ -Wall Ex1.cpp -o Ex1
Ex1.cpp: In function “int main()”:
Ex1.cpp:30:4: error: expected “,” or “;” before “Float_t”
Ex1.cpp:36:26: error: “qt” was not declared in this scope
```

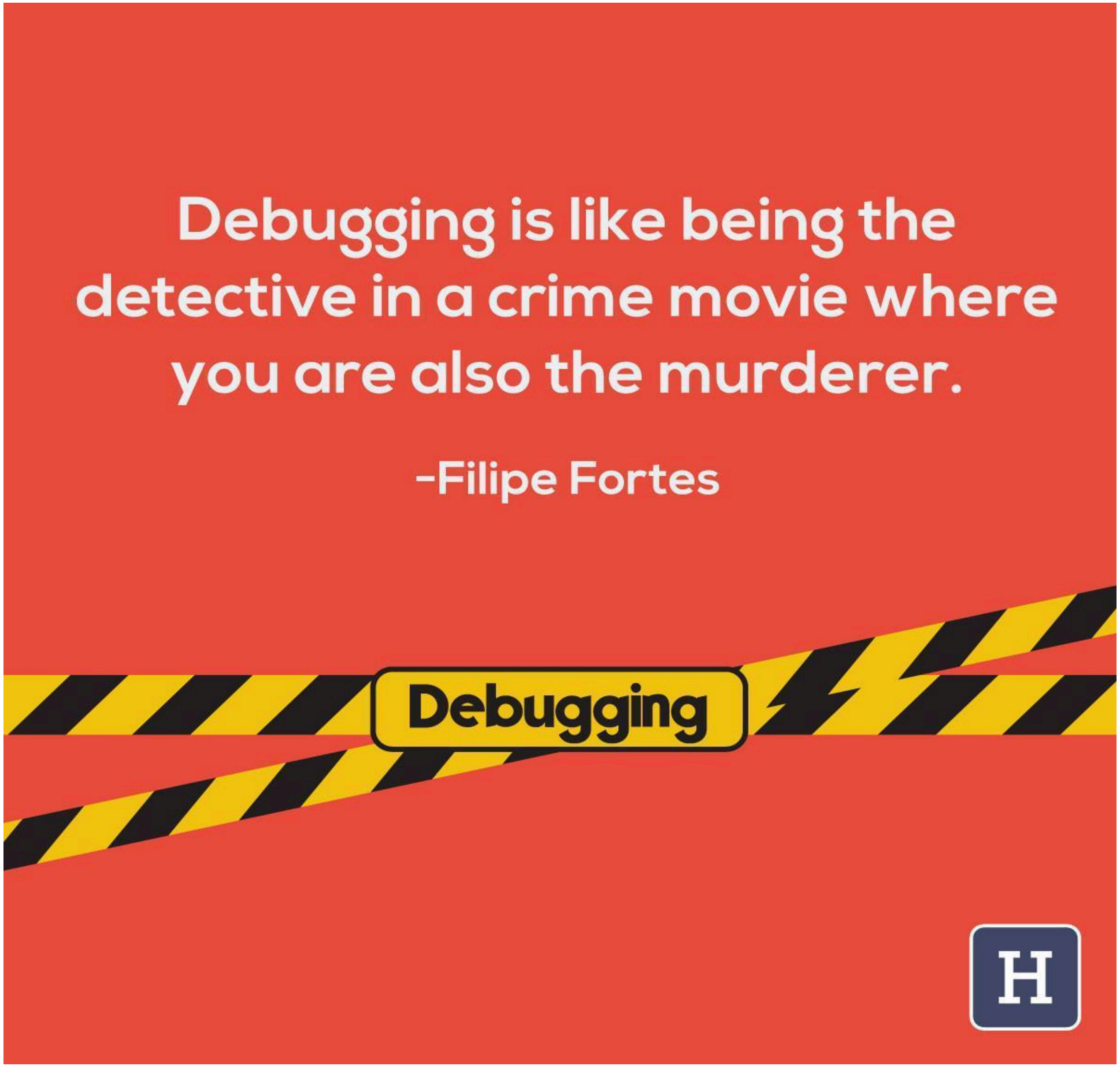
Debugging

First and more important rule of all:

READ CAREFULLY

<u>WARNINGS</u> AND <u>ERRORS</u>!

```
g++ -Wall Ex1.cpp -o Ex1
Ex1.cpp: In function “int main()”:
Ex1.cpp:30:4: error: expected “,” or “;” before “Float_t”
Ex1.cpp:36:26: error: “qt” was not declared in this scope
```


Debugging is like being the detective in a crime movie where you are also the murderer.
-Filipe Fortes
Debugging
H

Debugging, some tools

- strace
- ltrace
- ldd
- nm
- valgrind
- gdb/ddd

Debugging, strace

http://linux.die.net/man/1/strace

In the simplest case **strace** runs the specified *command* until it exits. It intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and its return value are printed.

Debugging, strace

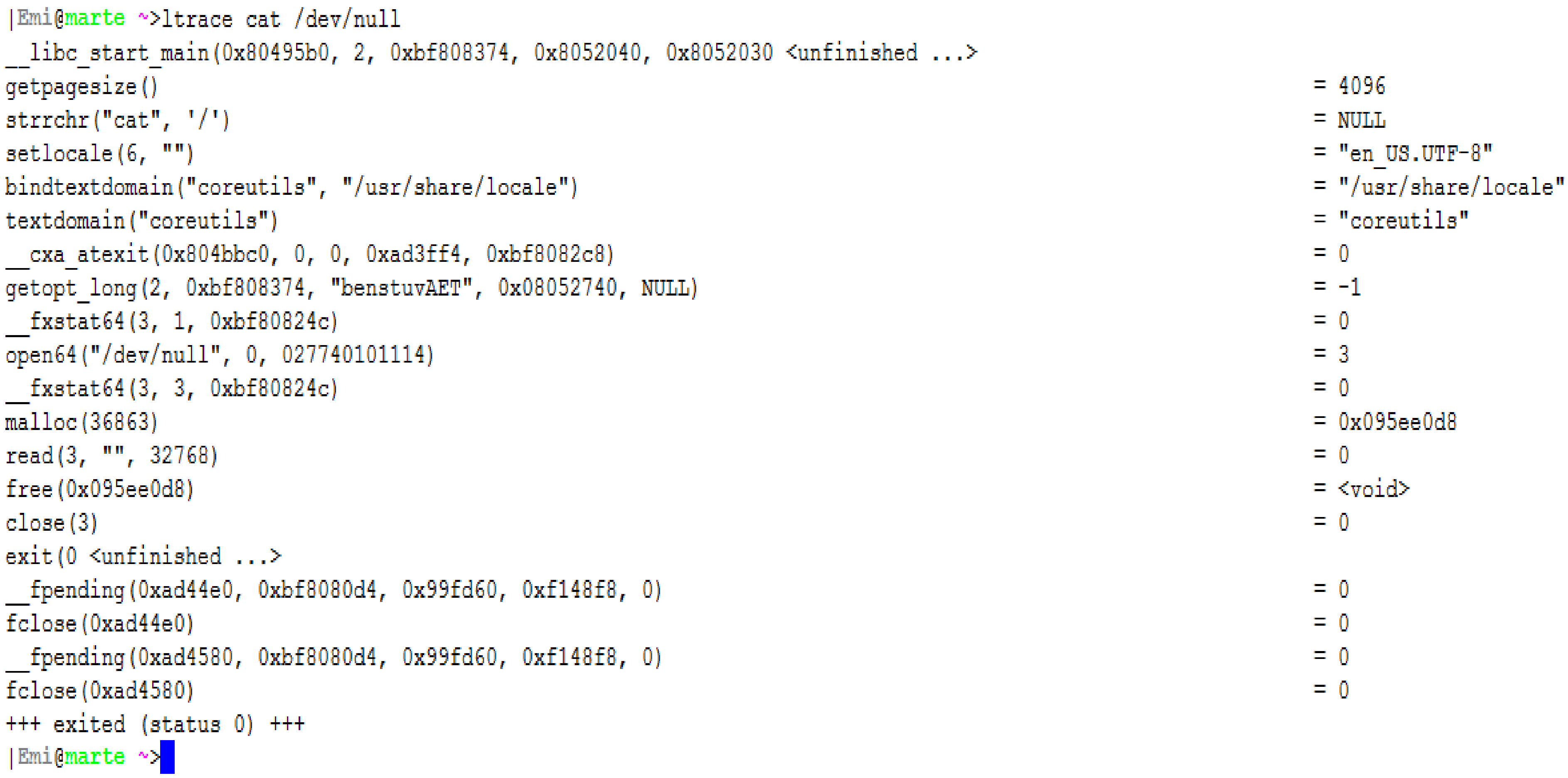
```
|Emi@marte ~>cat /dev/null
|Emi@marte ~>
|Emi@marte ~>strace cat /dev/null
execve("/bin/cat", ["cat", "/dev/null"], [/* 33 vars */]) = 0
brk(0)                                  = 0x16776000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2b5b4f81a000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2b5b4f81b000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=127827, ...}) = 0
mmap(NULL, 127827, PROT_READ, MAP_PRIVATE, 3, 0) = 0x2b5b4f81c000
close(3)                                = 0
open("/lib64/libc.so.6", O_RDONLY)      = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\332\1\3028\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1718232, ...}) = 0
mmap(0x38c2000000, 3498328, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x38c2000000
mprotect(0x38c214e000, 2093056, PROT_NONE) = 0
mmap(0x38c234d000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x14d000) = 0x38c234d000
mmap(0x38c2352000, 16728, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x38c2352000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2b5b4f83c000
arch_prctl(ARCH_SET_FS, 0x2b5b4f83c6e0) = 0
mprotect(0x38c234d000, 16384, PROT_READ) = 0
mprotect(0x38c1e1c000, 4096, PROT_READ) = 0
munmap(0x2b5b4f81c000, 127827)          = 0
brk(0)                                  = 0x16776000
brk(0x16797000)                         = 0x16797000
open("/usr/lib/locale/locale-archive", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=56430176, ...}) = 0
mmap(NULL, 56430176, PROT_READ, MAP_PRIVATE, 3, 0) = 0x2b5b4f83d000
close(3)                                = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
open("/dev/null", O_RDONLY)             = 3
fstat(3, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0
read(3, "", 4096)                       = 0
close(3)                                = 0
close(1)                                = 0
exit_group(0)                           = ?
|Emi@marte ~>
```

Debugging, ltrace

http://linux.die.net/man/1/ltrace

ltrace is a program that simply runs the specified *command* until it exits. It intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process. It can also intercept and print the system calls executed by the program.Its use is very similar to strace.

Debugging, ltrace



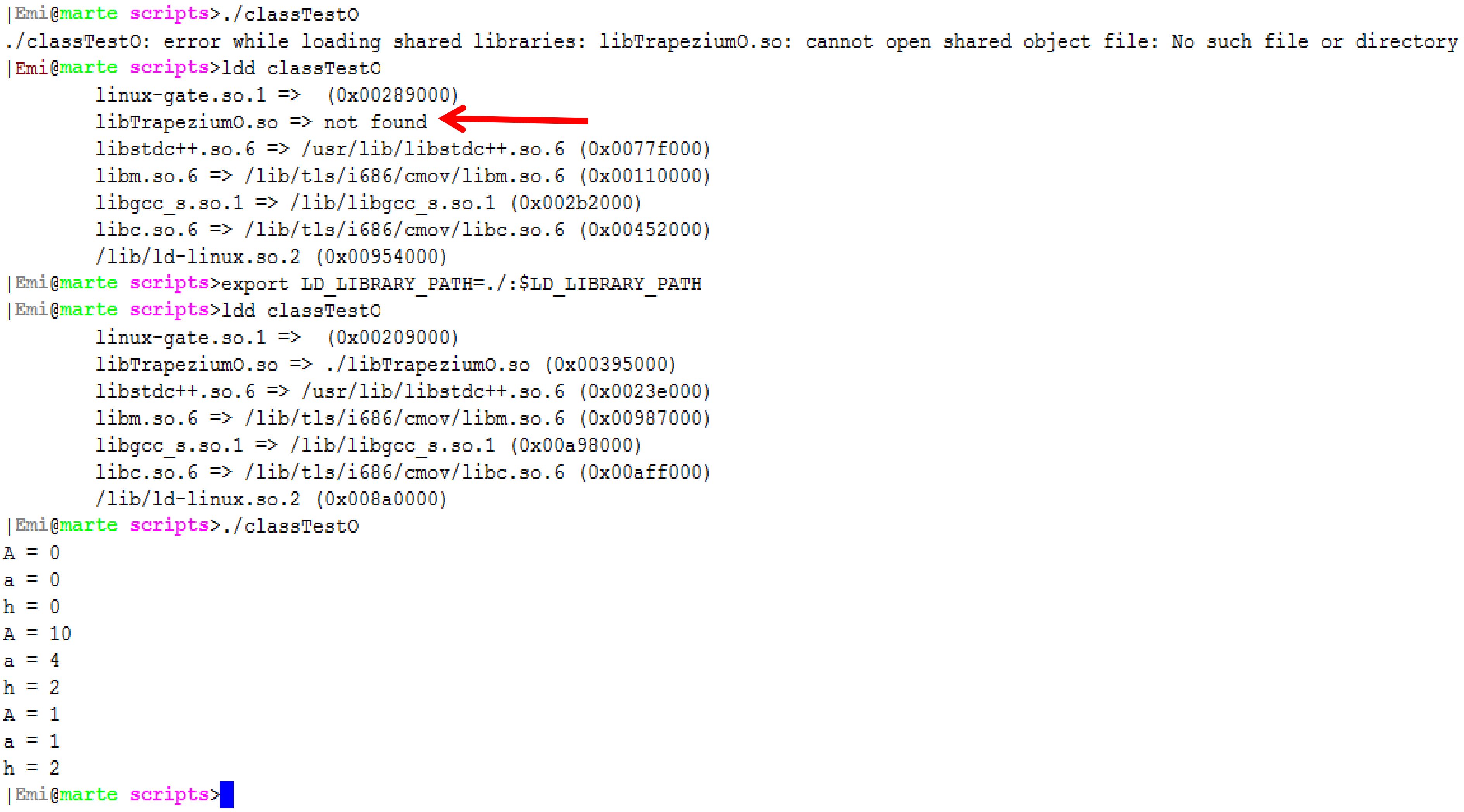
```
|Emi@marte ~>ltrace cat /dev/null
__libc_start_main(0x80495b0, 2, 0xbf808374, 0x8052040, 0x8052030 <unfinished ...>
getpagesize()                                                          = 4096
strrchr("cat", '/')                                                    = NULL
setlocale(6, "")                                                       = "en_US.UTF-8"
bindtextdomain("coreutils", "/usr/share/locale")                       = "/usr/share/locale"
textdomain("coreutils")                                                = "coreutils"
__cxa_atexit(0x804bbc0, 0, 0, 0xad3ff4, 0xbf8082c8)                    = 0
getopt_long(2, 0xbf808374, "benstuvAET", 0x08052740, NULL)             = -1
__fxstat64(3, 1, 0xbf80824c)                                           = 0
open64("/dev/null", 0, 027740101114)                                   = 3
__fxstat64(3, 3, 0xbf80824c)                                           = 0
malloc(36863)                                                          = 0x095ee0d8
read(3, "", 32768)                                                     = 0
free(0x095ee0d8)                                                       = <void>
close(3)                                                               = 0
exit(0 <unfinished ...>
__fpending(0xad44e0, 0xbf8080d4, 0x99fd60, 0xf148f8, 0)                = 0
fclose(0xad44e0)                                                       = 0
__fpending(0xad4580, 0xbf8080d4, 0x99fd60, 0xf148f8, 0)                = 0
fclose(0xad4580)                                                       = 0
+++ exited (status 0) +++
|Emi@marte ~>
```

Debugging, ldd

http://linux.die.net/man/1/ldd

ldd prints the shared libraries required by each program or shared library specified on the command line.

Debugging, ldd

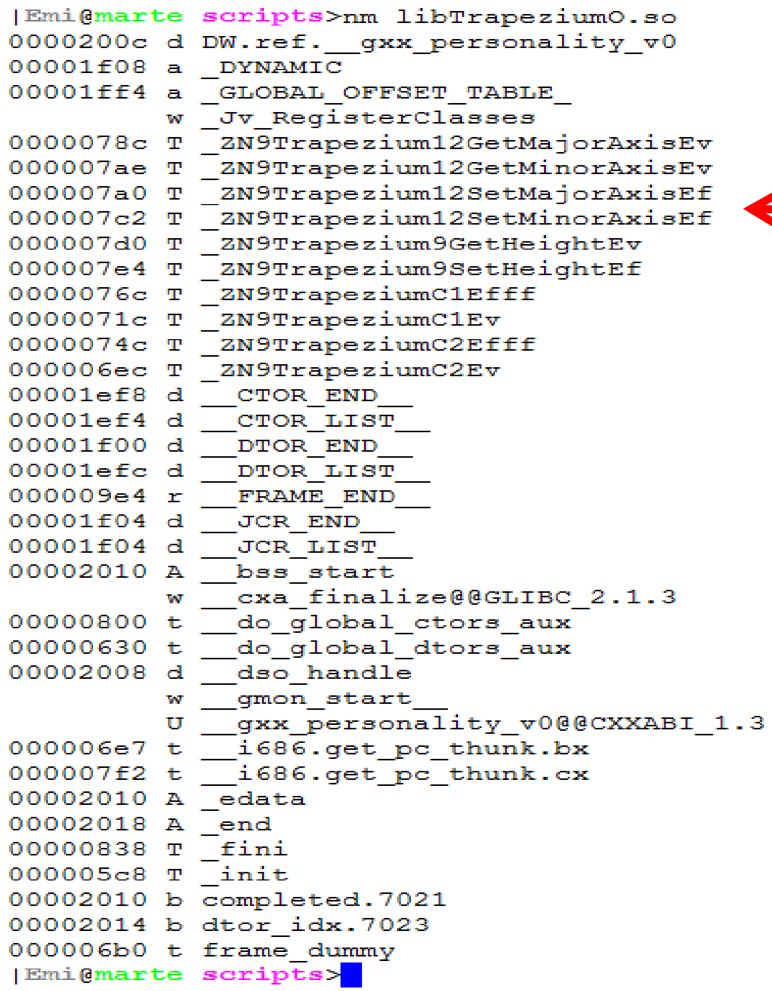


```
|Emi@marte scripts>./classTest0
./classTest0: error while loading shared libraries: libTrapezium0.so: cannot open shared object file: No such file or directory
|Emi@marte scripts>ldd classTest0
        linux-gate.so.1 =>  (0x00289000)
        libTrapezium0.so => not found
        libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x0077f000)
        libm.so.6 => /lib/tls/i686/cmov/libm.so.6 (0x00110000)
        libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x002b2000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00452000)
        /lib/ld-linux.so.2 (0x00954000)
|Emi@marte scripts>export LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH
|Emi@marte scripts>ldd classTest0
        linux-gate.so.1 =>  (0x00209000)
        libTrapezium0.so => ./libTrapezium0.so (0x00395000)
        libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x0023e000)
        libm.so.6 => /lib/tls/i686/cmov/libm.so.6 (0x00987000)
        libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x00a98000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00aff000)
        /lib/ld-linux.so.2 (0x008a0000)
|Emi@marte scripts>./classTest0
A = 0
a = 0
h = 0
A = 10
a = 4
h = 2
A = 1
a = 1
h = 2
|Emi@marte scripts>
```

Debugging, nm

http://linux.die.net/man/1/nm

nm lists the symbols from object files.



```
|Emi@marte scripts>nm libTrapezium0.so
0000200c d DW.ref.__gxx_personality_v0
00001f08 a _DYNAMIC
00001ff4 a _GLOBAL_OFFSET_TABLE_
         w _Jv_RegisterClasses
0000078c T _ZN9Trapezium12GetMajorAxisEv
000007ae T _ZN9Trapezium12GetMinorAxisEv
000007a0 T _ZN9Trapezium12SetMajorAxisEf
000007c2 T _ZN9Trapezium12SetMinorAxisEf
000007d0 T _ZN9Trapezium9GetHeightEv
000007e4 T _ZN9Trapezium9SetHeightEf
0000076c T _ZN9TrapeziumC1Efff
0000071c T _ZN9TrapeziumC1Ev
0000074c T _ZN9TrapeziumC2Efff
000006ec T _ZN9TrapeziumC2Ev
00001ef8 d __CTOR_END__
00001ef4 d __CTOR_LIST__
00001f00 d __DTOR_END__
00001efc d __DTOR_LIST__
000009e4 r __FRAME_END__
00001f04 d __JCR_END__
00001f04 d __JCR_LIST__
00002010 A __bss_start
         w __cxa_finalize@@GLIBC_2.1.3
00000800 t __do_global_ctors_aux
00000630 t __do_global_dtors_aux
00002008 d __dso_handle
         w __gmon_start__
         U __gxx_personality_v0@@CXXABI_1.3
000006e7 t __i686.get_pc_thunk.bx
000007f2 t __i686.get_pc_thunk.cx
00002010 A _edata
00002018 A _end
00000838 T _fini
000005c8 T _init
00002010 b completed.7021
00002014 b dtor_idx.7023
000006b0 t frame_dummy
|Emi@marte scripts>
```

class methods

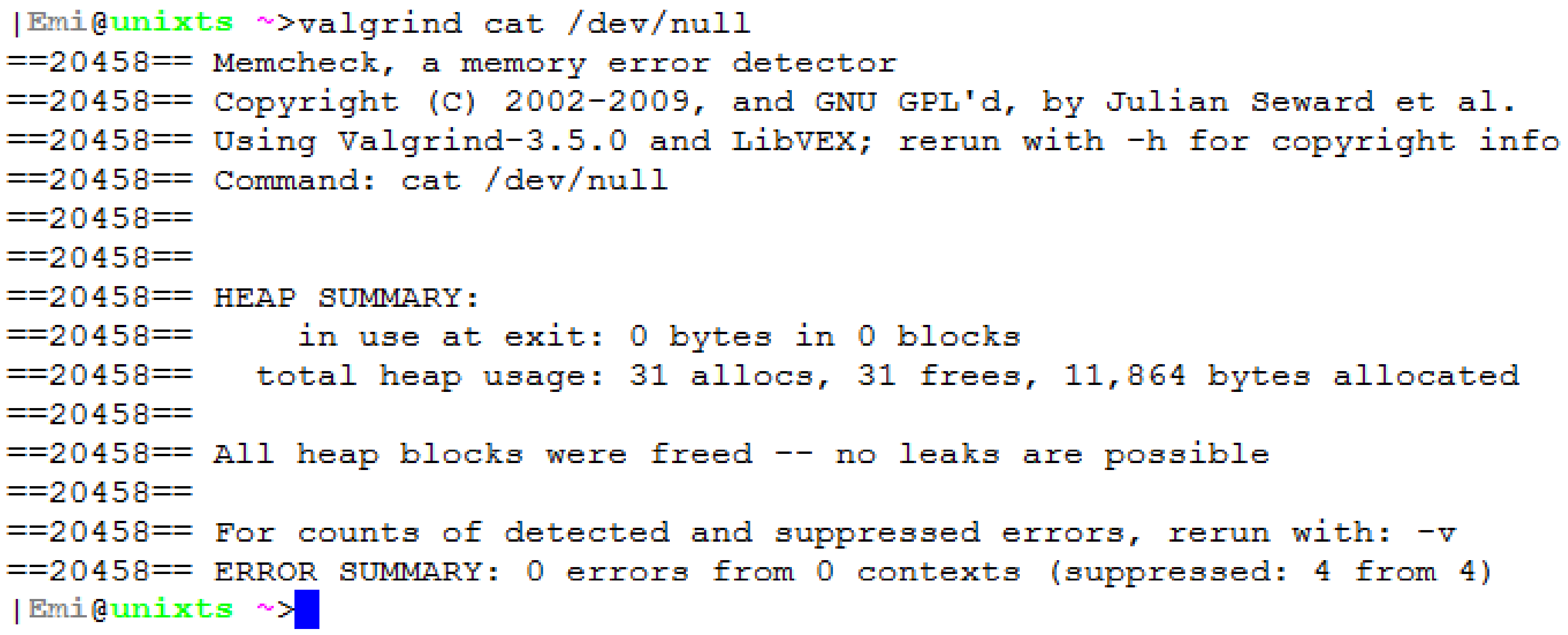
Debugging, valgrind

Not a standard installed program on ubuntu! ask your system managers if you need it!

http://valgrind.org/

Valgrind is a GPL'd system for debugging and profiling Linux programs. With Valgrind's tool suite you can automatically detect many memory management and threading bugs, avoiding hours of frustrating bug-hunting, making your programs more stable. You can also perform detailed profiling to help speed up your programs.

Debugging, valgrind



```
|Emi@unixts ~>valgrind cat /dev/null
==20458== Memcheck, a memory error detector
==20458== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==20458== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==20458== Command: cat /dev/null
==20458==
==20458==
==20458== HEAP SUMMARY:
==20458==     in use at exit: 0 bytes in 0 blocks
==20458==   total heap usage: 31 allocs, 31 frees, 11,864 bytes allocated
==20458==
==20458== All heap blocks were freed -- no leaks are possible
==20458==
==20458== For counts of detected and suppressed errors, rerun with: -v
==20458== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
|Emi@unixts ~>
```

Debugging, the ultimate debugger: GDB/DDD

http://linux.die.net/man/1/gdb

The purpose of a debugger such as GDB is to allow you to see what is going on "inside" another program while it executes-or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

Start your program, specifying anything that might affect its behavior.

- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

You can use GDB to debug programs written in C, C++, and Modula-2.

DDD is a graphic interface to GDB (and other debuggers):

http://www.gnu.org/software/ddd/

Debugging, GDB/DDD how to use it

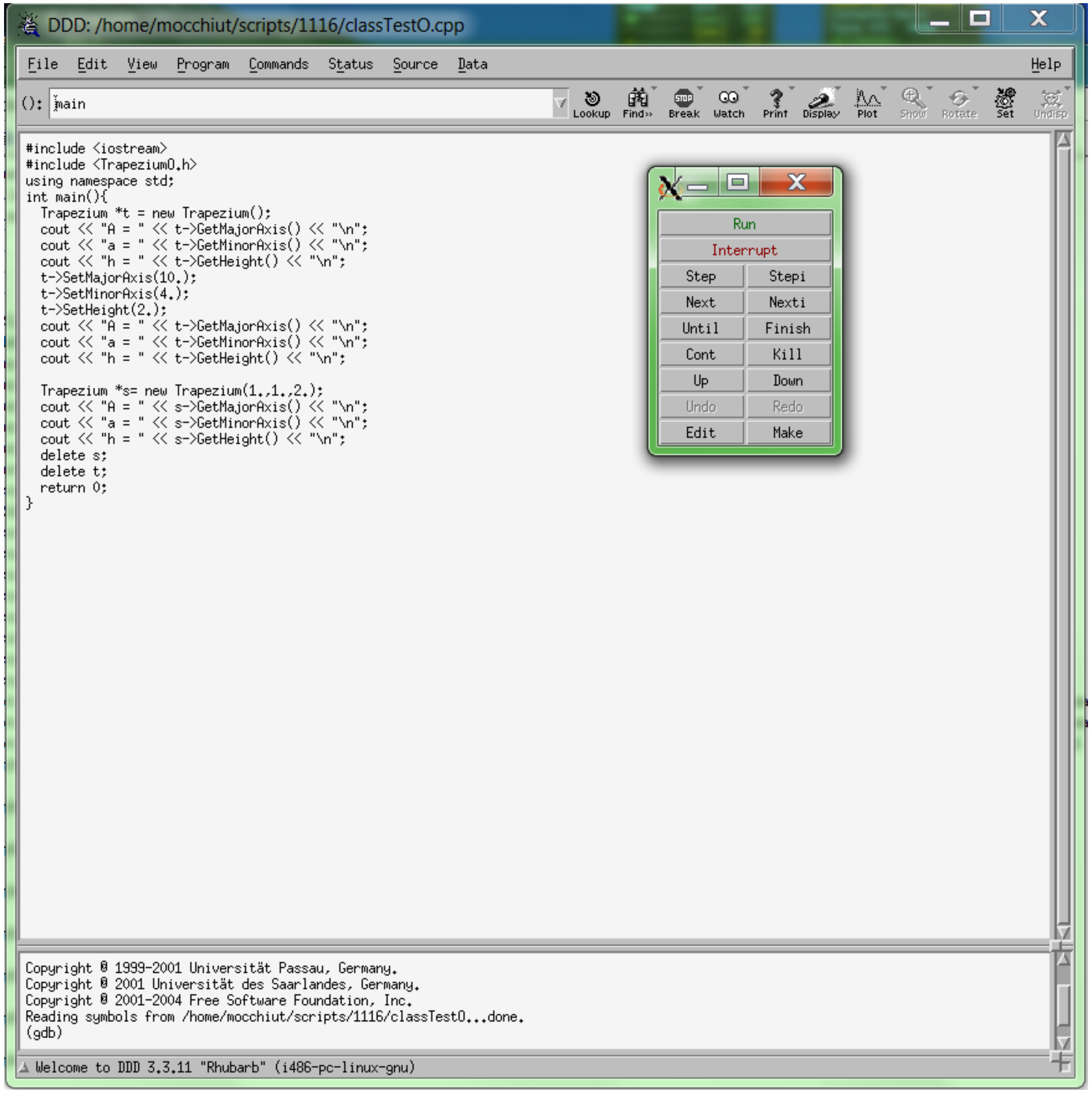
1. compile your executable and libraries with the option “-g”, for example:

```
g++ -Wall -g -I./ -L./ -fPIC –c Trapezium0.cpp
g++ -Wall -g -I./ -L./ -shared Trapezium0.o -o libTrapezium0.so
g++ -Wall -g -I./ -L./ -c classTest0.cpp -o classTest0.o
g++ -Wall -g -I./ -L./ libTrapezium0.so classTest0.o -o classTest0
```

2. run “ddd executable”

```
cd /home/mocchiut/scripts/classExample
ddd ./classTest
```

Debugging, GDB/DDD how to use it


DDD: /home/mocchiut/scripts/1116/classTestO.cpp
File Edit View Program Commands Status Source Data Help
(): main
Lookup Find» Break Watch Print Display Plot Show Rotate Set Undisp
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make

```
#include <iostream>
#include <TrapeziumO.h>
using namespace std;
int main(){
  Trapezium *t = new Trapezium();
  cout << "A = " << t->GetMajorAxis() << "\n";
  cout << "a = " << t->GetMinorAxis() << "\n";
  cout << "h = " << t->GetHeight() << "\n";
  t->SetMajorAxis(10.);
  t->SetMinorAxis(4.);
  t->SetHeight(2.);
  cout << "A = " << t->GetMajorAxis() << "\n";
  cout << "a = " << t->GetMinorAxis() << "\n";
  cout << "h = " << t->GetHeight() << "\n";

  Trapezium *s= new Trapezium(1.,1.,2.);
  cout << "A = " << s->GetMajorAxis() << "\n";
  cout << "a = " << s->GetMinorAxis() << "\n";
  cout << "h = " << s->GetHeight() << "\n";
  delete s;
  delete t;
  return 0;
}
```

```
Copyright © 1999-2001 Universität Passau, Germany.
Copyright © 2001 Universität des Saarlandes, Germany.
Copyright © 2001-2004 Free Software Foundation, Inc.
Reading symbols from /home/mocchiut/scripts/1116/classTestO...done.
(gdb)
```

Welcome to DDD 3.3.11 "Rhubarb" (i486-pc-linux-gnu)

make: makefiles

- Makefiles are a simple way to organize code compilation
- Complex and multiple compilation lines are coded in a text file (Makefile)
- Easy compilation from the prompt by giving the command "make"

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

int main() {
        cout << "ciccio" << endl;
        for (int i = 0; i < 10; i++) {
                cout << " i " << i << endl;
        }
        return 0;
}
bash> g++ -Wall test.cpp –o test
```

make: makefiles

```
Makefile:

hello: test.cpp
	g++ –Wall test.cpp -o test

bash> make hello
```

The basic makefile is composed of:

target: dependencies
[tab] system command

http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

http://mrbook.org/tutorials/make/

make: makefiles

```
Makefile:

hello: test.cpp
	g++ –Wall test.cpp -o test

bash> make hello
```

This is a “tab”, not 8 spaces!

make: makefiles

```
Makefile:

CC=g++
CFLAGS=-Wall
hello: test.cpp
	$(CC) $(CFLAGS) test.cpp -o test

bash> make hello
```

make: makefiles

```
Makefile:

CC=g++
CFLAGS=-Wall
hello: test.o
	$(CC) $(CFLAGS) test.o -o test

bash> make hello
```

By putting the object files - test.o - in the dependency list and in the rule, make knows it must first compile the .cpp versions individually, and then build the executable

make: makefiles

```
Makefile:

CC=g++
CFLAGS=-Wall
# if any test.h exists put it here
DEPS = test.h

%.o: %.cpp $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

hello: test.o
	$(CC) $(CFLAGS) test.o -o test

bash> make hello
```

commented lines start with #

rule that applies to all files ending in the .o suffix

The rule says that the .o file depends upon the .cpp version of the file and the .h files included in the DEPS macro. The rule then says that to generate the .o file, make needs to compile the .cpp file. The -o $@ says to put the output of the compilation in the file named on the left side of the :, the $< is the first item in the dependencies list

make: makefiles

```
Makefile:

CC=g++
CFLAGS=-Wall
# if any test.h exists put it here
DEPS = test.h
OBJ = test.o
%.o: %.cpp $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

hello: $(OBJ)
	$(CC) $(CFLAGS) $^ -o $@

bash> make hello
```

The $^ represents the right side of the :

make: cmake

• cmake is a command that generates the makefiles given a set of text files properly and simply formatted.
• it permits (and require) a good organization of files in directories and subdirectories.

http://www.cmake.org/

http://www.cmake.org/cmake-tutorial

http://www.elpauer.org/stuff/learning_cmake.pdf

Cmake: simplest example

- one executable: hello.cpp

```
#include <iostream>
using namespace std;

int main(){
        cout<<"hello world!\n";
        return 0;
}
```

```
bash# ls
hello.cpp
```

Cmake: simplest example

Suggestion: create a good directory tree, for example:

empty at the beginning

```
bash# ls -l
drwxr-x--- 2 mocchiut users 4.0K Apr 15 22:02 build/
drwxr-x--- 5 mocchiut users 4.0K Apr 15 22:01 installed/
drwxr-x--- 3 mocchiut users 4.0K Apr 15 21:48 trapezio/
```

source code

build : directory needed only for compilation of programs
installed : directory where project files (executables, libraries, headers,...) are installed
trapezio : directory containing the project source files only

Cmake: simplest example

Suggestion: create a good directory tree, for example:

```
bash# ls build
bash# ls installed
bash# ls trapezio
hello.cpp
bash#
```

build : directory needed only for compilation of programs
installed : directory where project files (executables, libraries, headers,...) are installed
trapezio : directory containing the project source files only

Cmake: simplest example

Step1: create the main cmake file in the project main directory

```
bash# cd trapezio
bash# gedit CMakeLists.txt
```

cmake files MUST be called “CMakeLists.txt”

```
cmake_minimum_required (VERSION 2.6)
project (Trapezio)
add_executable(hello hello.cpp)
```

done! you are ready to compile hello.cpp!!

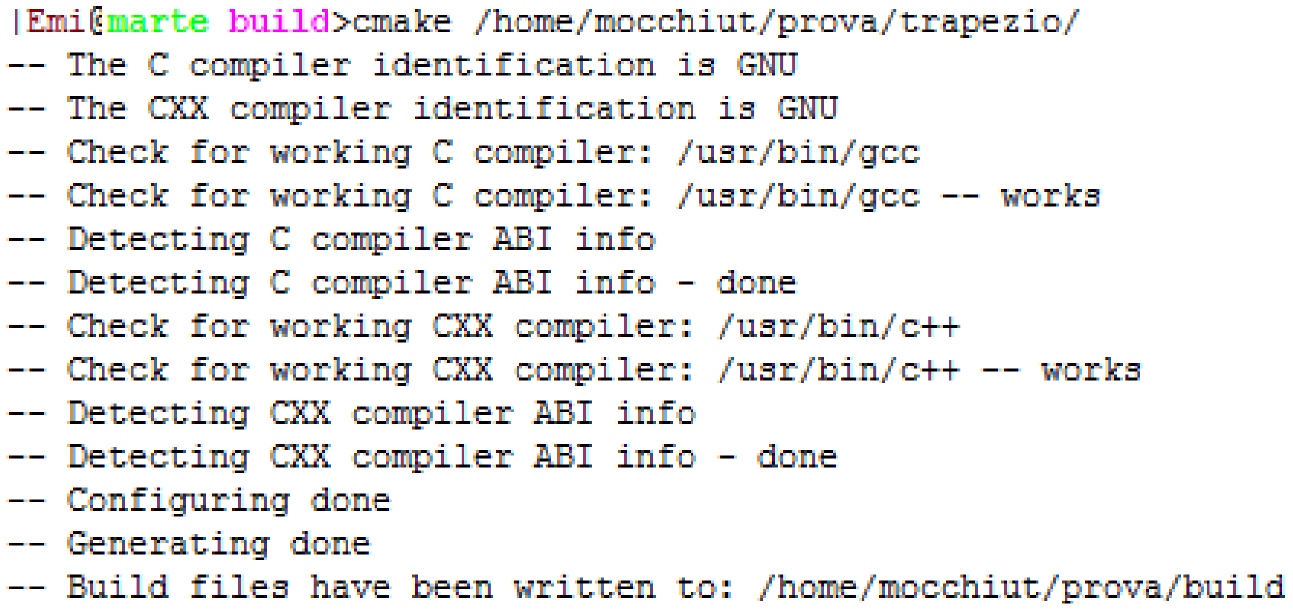
Cmake: simplest example

Step2: go to the compilation directory (it can be anywhere on the disk and it can be called as you prefer), we created “build”, so:

bash# cd ../

bash# cd build

bash# cmake /full/path/to/trapezio

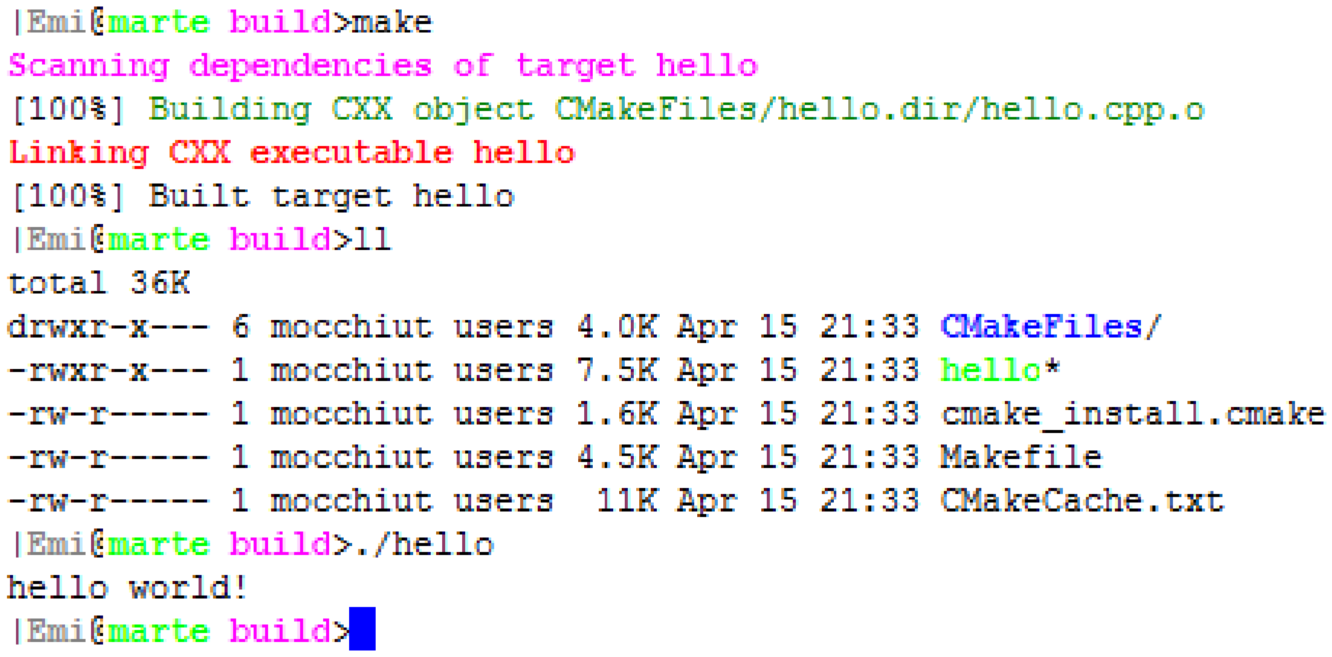


```
|Emi@marte build>cmake /home/mocchiut/prova/trapezio/
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/mocchiut/prova/build
```

bash# make

Cmake: simplest example

Step3: inside the compilation directory give the command:
bash# make



```
|Emi@marte build>make
Scanning dependencies of target hello
[100%] Building CXX object CMakeFiles/hello.dir/hello.cpp.o
Linking CXX executable hello
[100%] Built target hello
|Emi@marte build>ll
total 36K
drwxr-x--- 6 mocchiut users 4.0K Apr 15 21:33 CMakeFiles/
-rwxr-x--- 1 mocchiut users 7.5K Apr 15 21:33 hello*
-rw-r----- 1 mocchiut users 1.6K Apr 15 21:33 cmake_install.cmake
-rw-r----- 1 mocchiut users 4.5K Apr 15 21:33 Makefile
-rw-r----- 1 mocchiut users  11K Apr 15 21:33 CMakeCache.txt
|Emi@marte build>./hello
hello world!
|Emi@marte build>
```

Cmake: how to install files

Edit the CMakeLists.txt:

```
bash# cd trapezio
bash# gedit CMakeLists.txt
```

```
cmake_minimum_required (VERSION 2.6)
project (Trapezio)
add_executable(hello hello.cpp)

install (TARGETS hello DESTINATION bin)
```

add the red line and go back to the compilation directory

Cmake: how to install

Go to the compilation directory:

```
bash# cd ../
bash# cd build
bash# cmake -DCMAKE_INSTALL_PREFIX=/full/path/to/installed/ /full/path/to/trapezio

bash# make all install

bash# ls -R /full/path/to/installed/
/full/path/to/installed/:
bin/

/full/path/to/installed/bin:
hello*
```

Cmake: how to install

NB: to fully exploit the potentiality of installation edit the file ".bashrc" in your home directory and add something like:

```
export PATH=/full/path/to/installed/bin:$PATH
export LD_LIBRARY_PATH=/full/path/to/installed/lib:$LD_LIBRARY_PATH
```

Cmake: libraries how to

Assume to have:

- a class made of header and implementation files
- a program using that class
- another independent program

for example:

```
bash# cd trapezio
bash# ls
hello.cpp classTestI.cpp Polygon.cpp Polygon.h
```

Polygon.h is included in classTestI.cpp

we want to create a shared library, compile the two programs and properly install everything.

Cmake: libraries how to

CMakeLists.txt will be:

```
cmake_minimum_required (VERSION 2.6)
project (Trapezio)

include_directories("${PROJECT_SOURCE_DIR}")

add_executable(hello hello.cpp)

add_executable(classTestI classTestI.cpp)
target_link_libraries (classTestI Polygon)

add_library(Polygon SHARED Polygon.cpp)

install (FILES Polygon.h DESTINATION include)
install (TARGETS Polygon DESTINATION lib)

install (TARGETS hello DESTINATION bin)
install (TARGETS classTestI DESTINATION bin)
```

Cmake: libraries how to

CMakeLists.txt will be:

```
cmake_minimum_required (VERSION 2.6)
project (Trapezio)

include_directories("${PROJECT_SOURCE_DIR}")
```

look for headers in the project directory

```
add_executable(hello hello.cpp)

add_executable(classTestI classTestI.cpp)
target_link_libraries (classTestI Polygon)

add_library(Polygon SHARED Polygon.cpp)

install (FILES Polygon.h DESTINATION include)
install (TARGETS Polygon DESTINATION lib)

install (TARGETS hello DESTINATION bin)
install (TARGETS classTestI DESTINATION bin)
```

Cmake: libraries how to

CMakeLists.txt will be:

```
cmake_minimum_required (VERSION 2.6)
project (Trapezio)

include_directories("${PROJECT_SOURCE_DIR}")

add_executable(hello hello.cpp)

add_executable(classTestI classTestI.cpp)
target_link_libraries (classTestI Polygon)

add_library(Polygon SHARED Polygon.cpp)

install (FILES Polygon.h DESTINATION include)
install (TARGETS Polygon DESTINATION lib)

install (TARGETS hello DESTINATION bin)
install (TARGETS classTestI DESTINATION bin)
```

the executable classTestI will need to be linked to a library called libPolygon.so

Cmake: libraries how to

CMakeLists.txt will be:

```
cmake_minimum_required (VERSION 2.6)
project (Trapezio)

include_directories("${PROJECT_SOURCE_DIR}")

add_executable(hello hello.cpp)

add_executable(classTestI classTestI.cpp)
target_link_libraries (classTestI Polygon)

add_library(Polygon SHARED Polygon.cpp)

install (FILES Polygon.h DESTINATION include)
install (TARGETS Polygon DESTINATION lib)

install (TARGETS hello DESTINATION bin)
install (TARGETS classTestI DESTINATION bin)
```

create a shared library called libPolygon.so
from file Polygon.cpp

Cmake: how to install

Go to the compilation directory:

```
bash# cd ../
bash# cd build
bash# cmake -DCMAKE_INSTALL_PREFIX=/full/path/to/installed/ /full/path/to/trapezio
bash# make all install
bash# ls -R /full/path/to/installed/
/full/path/to/installed/:
bin/ lib/ include/

/full/path/to/installed/bin:
classTestI* hello*

/full/path/to/installed/lib:
libPolygon.so

/full/path/to/installed/include:
Polygon.h
```

Standard Template Library (STL)

Templates

http://www.codeproject.com/Articles/257589/An-Idiots-Guide-to-Cplusplus-Templates-Part-1

Suppose you write a function that prints an input number multiplied by two:

```
void PrintTwice(int data) {
  cout << "Twice is: " << data * 2 << endl;
}
```

Which can be called passing an int:

```
PrintTwice(120); // 240
```

Templates

Now, if you want to print double of a double, you would overload this function as:

```
void PrintTwice(double data) {
   cout << "Twice is: " << data * 2 <<
  endl;
}
```

Interestingly, class type ostream (the *type* of cout object) has multiple overloads for operator << for all basic data-types. Therefore, same/similar code works for both int and double, and no change is required for our PrintTwice overloads – yes, we just *copy-pasted* it.

Templates

```
void PrintTwice(int data) {
  cout << "Twice is: " << data * 2 << endl;
}
void PrintTwice(double data) {
  cout << "Twice is: " << data * 2 << endl;
}
```

This is one of the many situations where we can utilize the groovy feature provided by the C++ language: Templates!

Templates are of two types:

- Function Templates
- Class Templates

C++ templates is a programming model that allows plugging-in of any data-type to the code (templated code). Without template, you would need to replicate same code all over again and again, for all required data-types. And obviously, as said before, it requires code maintenance.

Templates

Here is the *templated* function PrintTwice:

```
template<class TYPE>
void PrintTwice(TYPE data) {
  cout<<"Twice: " << data * 2 << endl;
}
```

The first line of code:

template<class TYPE>

tells the compiler that this is a *function-template.* The actual meaning of TYPE would be deduced by compiler depending on the argument passed to this function. Here, the name, TYPE is known as **template type parameter.**

Templates

```
PrintTwice(124);
```

TYPE would be replaced by compiler as int, and compiler would **instantiate** this template-function as:

```
void PrintTwice(int data) {
  cout<<"Twice: " << data * 2 << endl;
}
```

And, if we call this function as:

```
PrintTwice(4.5547);
```

It would instantiate another function as:

```
void PrintTwice(double data) {
  cout<<"Twice: " << data * 2 << endl;
}
```

It means, in your program, if you call PrintTwice function with int and double parameter types, **two** instances of this function would be generated by compiler:

```
void PrintTwice(int data) { ... }
void PrintTwice(double data) { ... }
```

Yes, the code is duplicated. But these two overloads are instantiated by the ***compiler and not by the programmer***. The true benefit is that you need not to do *copy-pasting* the same code, or to manually maintain the code for different data-types, or to write up a new overload for new data-type that arrives later. You would just provide a **template** of a function, and rest would be managed by compiler.

Templates

```
template<class TYPE>
TYPE Add(TYPE n1, TYPE n2) {
   TYPE result;
   result = n1 + n2;
   return result;
}
```

Under the assumption that class TYPE:

- is having a default constructor (so that `TYPE result;` is valid)
- supports the usage of operator + (so that `n1+n2` is valid)
- has an *accessible* copy/move-constructor (so that return statement succeeds)

Standard Template Library

http://cs.stmarys.ca/~porter/csc/ref/stl/

What is the STL?

It's a sophisticated and powerful library of template classes and template functions that implement many common data structures and algorithms, and forms part of the C++ Standard Library.

Why should a C++ programmer be interested in the STL?

Because the STL embodies the concept of *reusable software components*, and provides off-the-shelf solutions to a wide variety of programming problems. It is also *extensible*, in the sense that any programmer can write new software (containers and algorithms, for example), that "fit in" to the STL and work with the already-existing parts of the STL, provided the programmer follows the appropriate design guidelines.

What is the design philosophy of the STL?

The STL exemplifies *generic programming* rather than *object-oriented programming*, and derives its power and flexibility from the use of templates, rather than inheritance and polymorphism. It also avoids new and delete for memory management in favor of *allocators* for storage allocation and deallocation. The STL also provides *performance guarantees*, i.e., its specification requires that the containers and algorithms be implemented in such a way that a user can be confident of optimal runtime performance independent of the STL implementation being used.

Standard Template Library

http://cs.stmarys.ca/~porter/csc/ref/stl/ – tutorial introduction

The STL has things called *containers*, *iterators* and *algorithms*. Our purpose here will be to see how these three things come together in what might be called "typical STL fashion".

Fortunately, we can begin by drawing some analogies between STL containers, iterators and algorithms and things you already know about.

STL – analogies

- We begin by noting that, at a superficial level at least, any STL *container* is analogous to an *array*, in that it is something that allows you to store and retrieve elements. All STL containers are implemented by *template* classes, For example, the STL has a container called vector, which the most array-like of all the STL containers, and in in fact it is often billed as a "better array".

containers $\longrightarrow$ arrays

STL – analogies

- Your C++ experience will have provided you with an opportunity to use *pointers* to "point at" and manipulate array elements. STL*iterators* are used to "point at" and manipulate STL container elements in a manner quite analogous to the way in which pointers are used in the array context. But ... each STL container class will have its own kind of iterator, one which is most appropriate for that particular container.

STLiterators $\Rightarrow$ pointers

STL – analogies

- And *algorithms* ... well, you know what they are. Often they come in the form of stand-alone or "free" functions which usually take in some data via a parameter list and perform some task and/or return one or more computed values. And all of this is true of STL algorithms as well. But ... where STL algorithms get their additional power and flexibility is from the fact that many of their parameters are *STL iterators* rather than STL containers. Since these iterators can point at the elements of many different kinds of STL containers this means that STL algorithms have a "built-in" ability to deal with a variety of containers containing a variety of element types. In other words, an algorithm can work on the elements of a container without knowing or caring what kind of container it is.

algorithms $\longrightarrow$ functions

STL – vector example

```
int a[10];
```

declaration of an ordinary C++ “array of integers” of size 10, a place to store 10 values of type *`int`*.

Using STL we can do:

```
#include <vector>

...

vector<int> v(10);
```

declaration of a STL-style “vector of integers” of size 10, a place to store 10 values of type *`int`*.

So... what is the difference?

STL – `vector` example, index type

The elements of `v` are `v[0], v[1], ... v[9],` just as the elements of `a` are `a[0], a[1], ... a[9],` and both are used in the same way. But even here there is a “hidden” and potentially confusing difference to be aware of in the *type* of the index variable.

To output the values in the array a one would normally use a loop idiom like the following:

```
for (int i=0; i<10; i++) cout << a[i];
```

With STL we must write:

```
for (vector<int>::size_type i=0; i<10; i++) cout
   << v[i];
```

The reason for this is that the *`int`* data type has values (all the negative ones) that cannot possibly be valid indices for a vector. Thus the vector class would like to be sure that we always use non-negative index values when accessing vector components and provides for us an alias called size_type (an unsigned integer type) for just this purpose.

STL – `vector` example, self-knowledge

A vector, unlike an array, has self-knowledge: it knows how big it is.

When outputting values for an array, the programmer needs to know the size of the array. Another advantage of a vector over an array is that (because the *vector* is a *class object*) the vector knows its own size.

Hence we can (and should) write:

```
for (vector<int>::size_type i=0; i<v.size(); i++)
  cout << v[i];
```

which will work for any vector of whatever size.

We no longer need to know and remember that the size of the vector v is 10, in this example.

STL – `vector` example, initialization

Unlike a, v is a *class object*, and we have used one of several constructors from the vector container class to create v.

Among other things, this means that each element of v is guaranteed to be initialized with the default value of the component type (0 here, since the component type is *int*).

On the other hand, the elements of a may or may not be initialized to 0, depending on where the declaration is located in the program, for example.

STL – `vector` example, variable size

Once declared as above, the size of a is fixed and cannot be altered later during the program execution.

But the vector class contains a *push_back()* member function which can be used like this

```
v.push_back(17);
```

to add the value 17 (for example) to the end of the vector.

After execution of this statement the size of the vector v has increased to 11 and v[10] contains the value 17 (remember that the first 10 values of v are all 0, and are found in the locations with indices in the range 0,...,9).

This ability of a vector to “grow” in order to accommodate the addition of new values to its “back end” is one of the major advantages of vectors over arrays.

STL – `vector` example, init with an array

Although declaring an empty array, as in

```
int a[0];
```

makes no sense, since nothing can ever be stored in it, declaring an empty vector and then adding some values makes perfect sense:

```
vector<int> v; //construct an empty vector to hold some integers
v.push_back(3); // v now contains 3 in v[0]
v.push_back(6); // v now contains 3 in v[0], 6 in v[1]
v.push_back(4); // v now contains 3 in v[0], 6 in v[1], 4 in v[2]
```

This is clearly a rather laborious and inconvenient way to get values into a vector. Since an array containing the same values can be initialized with the statement

```
int a[3] = {3, 6, 4};
```

a useful way of combining the old and the new to initialize a vector with a sequence of specific values is illustrated by the following two lines of code:

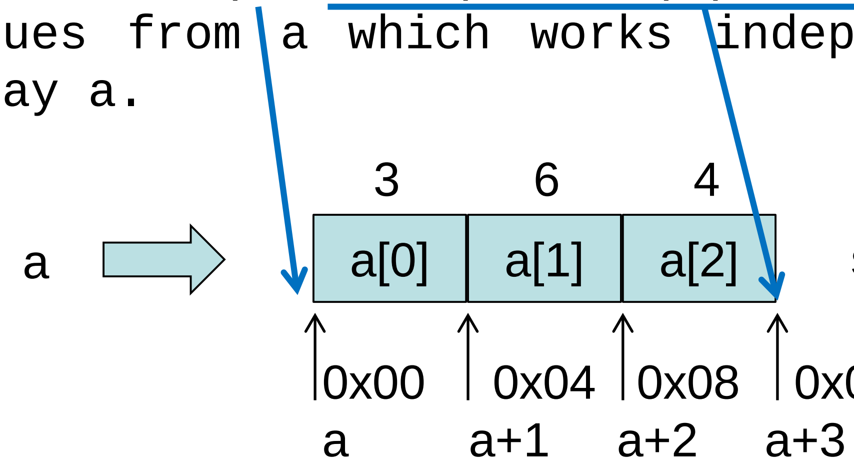
```
int a[3] = {3, 6, 4}; //initialize array a
vector<int> v(a, a + (sizeof(a)/sizeof(int))); //construct v with values
   from a which works independently of the size of the array a.
```

This vector definition (declaration with initialization) requires a few words of explanation. We are using another constructor from the vector class, this time the one that takes two input parameters which are "iterators" (or in this case, ordinary pointers acting as iterators) pointing at the beginning element, and "one-past-the-last" element of a range of values to be placed in the newly constructed vector.

In this case **the first parameter** (a), **is a pointer to the first element of the array** a, and **the second parameter** (`a+(sizeof(a)/sizeof(int))`) **is a pointer to the "first position beyond the last element" of the array** a. This notion of a "range" of elements being determined by a pair of iterators (or pointers) pointing to the first element and "one-past-the-last" element (as opposed to the last element itself) of a sequence of values permeates the STL and is something beginning STL programmers need to get a handle on right up front.

STL – vector example, init with an array

```
int a[3] = {3, 6, 4}; //initialize array a
vector<int> v(a, a + (sizeof(a)/sizeof(int)); //construct v with
   values from a which works independently of the size of the
   array a.
```



sizeof(int) = 4 bytes
sizeof(a) = 4 bytes times 3 = 12 bytes
so `a+(sizeof(a)/sizeof(int)=3`

pointers to array elements

In this case **the first parameter** (a), **is a pointer to the first element of the array** a, and **the second parameter** (`a+(sizeof(a)/sizeof(int))`) **is a pointer to the "first position beyond the last element" of the array** a. This notion of a "range" of elements being determined by a pair of iterators (or pointers) pointing to the first element and "one-past-the-last" element (as opposed to the last element itself) of a sequence of values permeates the STL and is something beginning STL programmers need to get a handle on right up front.

STL – `vector` example, iterators

The STL vector class provides an iterator which can be used to access the component values of any vector in the same way that ordinary pointers are used to access array elements.

The member function ***begin()*** returns an iterator object **pointing to the first component of the vector** on which it is invoked and the member function ***end()*** returns an iterator object **pointing to one-past-the-last component**.

The fact that it is one-past-the-last and not the last makes it convenient when writing a loop to process all vector elements, as in the following example.

```
int a[3] = {3, 6, 4};
vector<int> v(a, a+(sizeof(a)/sizeof(int)));
for (vector<int>::iterator p=v.begin(); p!=v.end();
  p++) cout << *p << " "; //output all values of v
```

Note that the iterator object p is incremented in the same way that an ordinary array pointer would be incremented, and that `*p` is used to dereference p and get access to the value to which p is pointing.

STL – `vector` example, algorithms

One particularly nice feature of the STL is that it provides a large collection of algorithms that perform many of the common programming tasks that programmers frequently need to have done in their programs.

Most of those algorithms are available from the `<algorithm>` header, but a few are located in the `<numeric>` header.

To quickly get a sense of how this works, look at this code

```
#include <algorithm>

...

sort(v.begin(), v.end());
```

which shows the STL sort algorithm being used to sort the values in a vector. By default this algorithm sorts in ascending order and will work for a vector of any items whatsoever, as long as operator `<()` is defined for the components of v, so that the sort algorithm knows what it means for one component to precede another.

STL – `vector` example, assignements and comparisons

A number of other operations are available for vectors that that have no counterpart when you are working with arrays.

For example, it is possible to assign one vector to another, as in

```
v1 = v2; // let v1 be (3,6,4), v2 is set to be
   (3,6,4) same as v1 (you need a loop using
   arrays!)
```

and the relational operators also work with vectors, so that you can make tests like

```
if (v1 == v2) ...
if (v1 < v2) ...
```

provided you know what `v1 (operator) v2` means.

Documentation, doxygen

Doxygen

Not a standard installed program on ubuntu! ask your system managers if you need it!

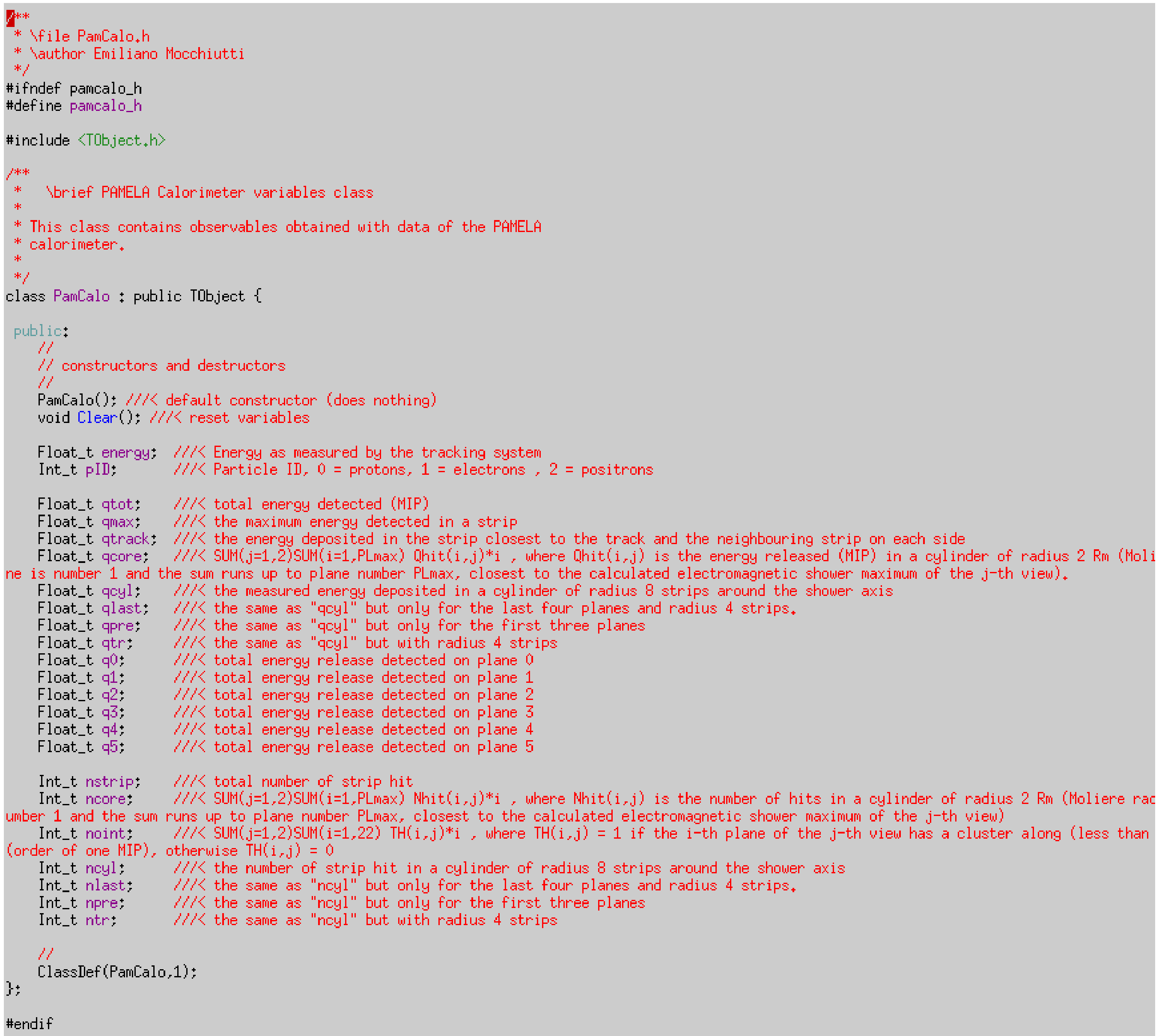
http://www.stack.nl/~dimitri/doxygen/

Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D.

Doxygen

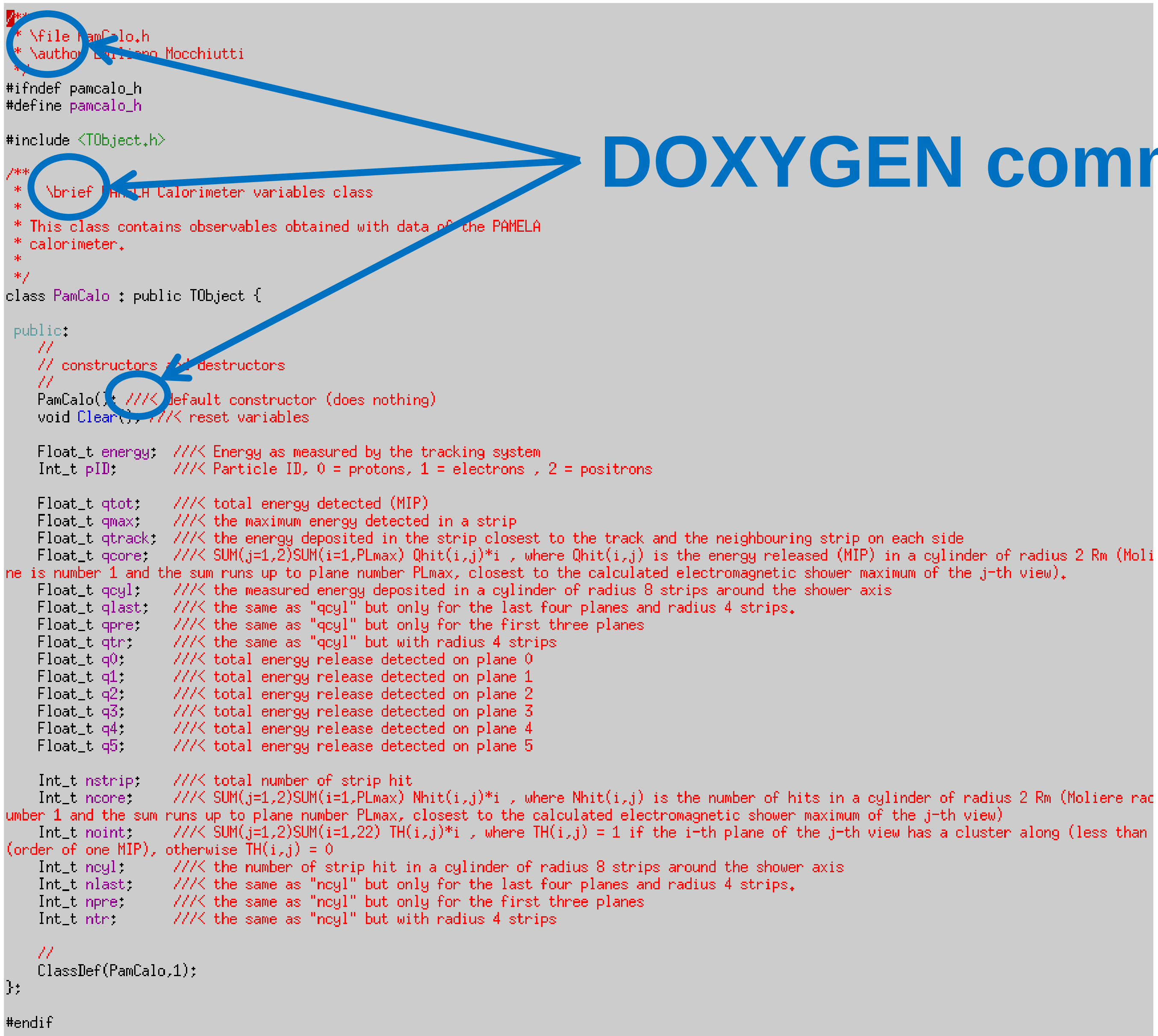
Doxygen can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in Latex) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The **documentation is extracted directly from the sources**, which makes it much easier to keep the documentation consistent with the source code.

Doxygen, source commenting



```
/**
 * \file PamCalo.h
 * \author Emiliano Mocchiutti
 */
#ifndef pamcalo_h
#define pamcalo_h

#include <TObject.h>

/**
 *   \brief PAMELA Calorimeter variables class
 *
 * This class contains observables obtained with data of the PAMELA
 * calorimeter.
 *
 */
class PamCalo : public TObject {

 public:
    //
    // constructors and destructors
    //
    PamCalo(); ///< default constructor (does nothing)
    void Clear(); ///< reset variables

    Float_t energy;  ///< Energy as measured by the tracking system
    Int_t pID;       ///< Particle ID, 0 = protons, 1 = electrons , 2 = positrons

    Float_t qtot;    ///< total energy detected (MIP)
    Float_t qmax;    ///< the maximum energy detected in a strip
    Float_t qtrack;  ///< the energy deposited in the strip closest to the track and the neighbouring strip on each side
    Float_t qcore;   ///< SUM(j=1,2)SUM(i=1,PLmax) Qhit(i,j)*i , where Qhit(i,j) is the energy released (MIP) in a cylinder of radius 2 Rm (Moli
ne is number 1 and the sum runs up to plane number PLmax, closest to the calculated electromagnetic shower maximum of the j-th view).
    Float_t qcyl;    ///< the measured energy deposited in a cylinder of radius 8 strips around the shower axis
    Float_t qlast;   ///< the same as "qcyl" but only for the last four planes and radius 4 strips.
    Float_t qpre;    ///< the same as "qcyl" but only for the first three planes
    Float_t qtr;     ///< the same as "qcyl" but with radius 4 strips
    Float_t q0;      ///< total energy release detected on plane 0
    Float_t q1;      ///< total energy release detected on plane 1
    Float_t q2;      ///< total energy release detected on plane 2
    Float_t q3;      ///< total energy release detected on plane 3
    Float_t q4;      ///< total energy release detected on plane 4
    Float_t q5;      ///< total energy release detected on plane 5

    Int_t nstrip;    ///< total number of strip hit
    Int_t ncore;     ///< SUM(j=1,2)SUM(i=1,PLmax) Nhit(i,j)*i , where Nhit(i,j) is the number of hits in a cylinder of radius 2 Rm (Moliere rac
umber 1 and the sum runs up to plane number PLmax, closest to the calculated electromagnetic shower maximum of the j-th view)
    Int_t noint;     ///< SUM(j=1,2)SUM(i=1,22) TH(i,j)*i , where TH(i,j) = 1 if the i-th plane of the j-th view has a cluster along (less than
(order of one MIP), otherwise TH(i,j) = 0
    Int_t ncyl;      ///< the number of strip hit in a cylinder of radius 8 strips around the shower axis
    Int_t nlast;     ///< the same as "ncyl" but only for the last four planes and radius 4 strips.
    Int_t npre;      ///< the same as "ncyl" but only for the first three planes
    Int_t ntr;       ///< the same as "ncyl" but with radius 4 strips

    //
    ClassDef(PamCalo,1);
};

#endif
```

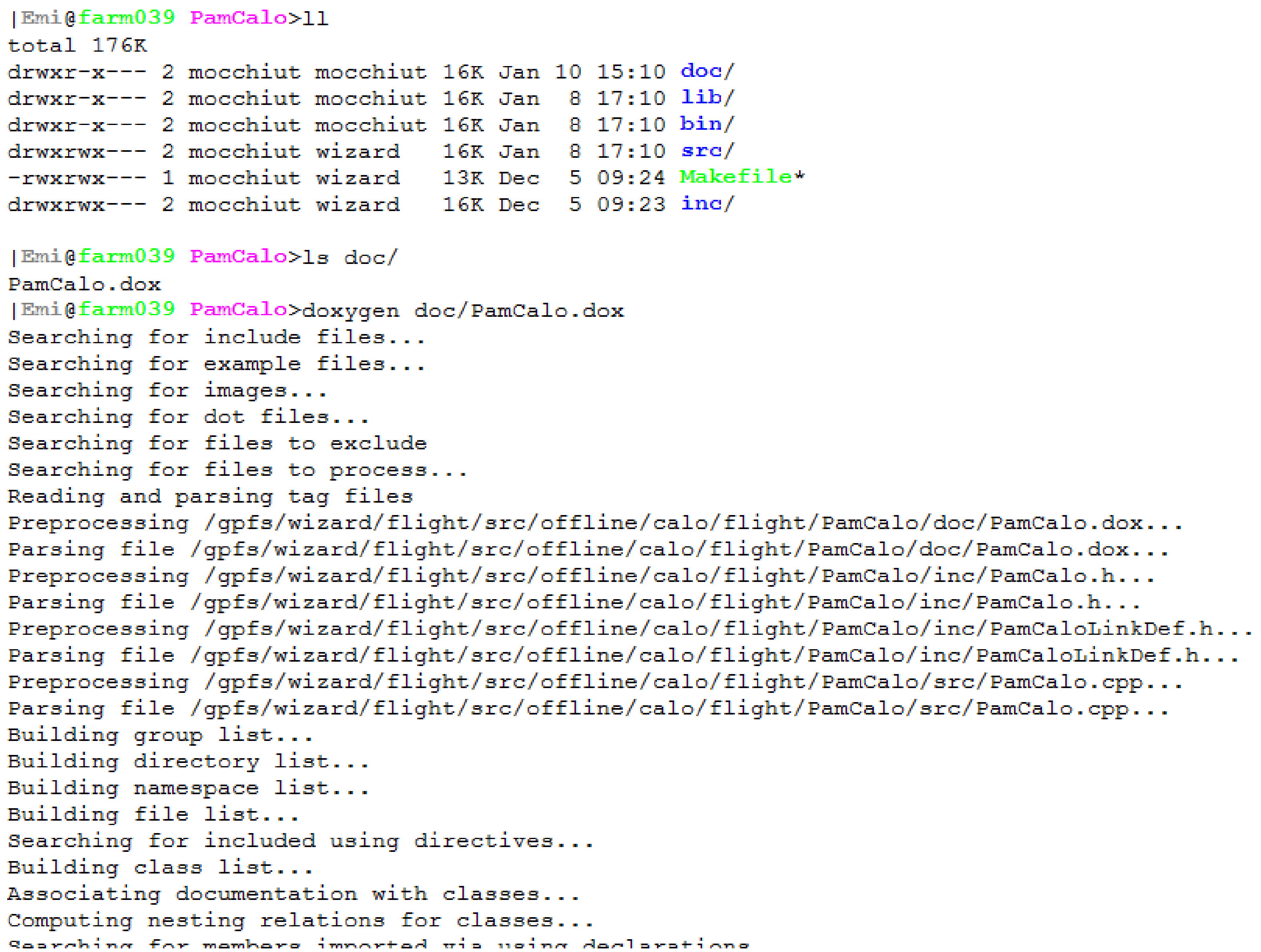
Doxygen, source commenting



```
/**
 * \file PamCalo.h
 * \author Emiliano Mocchiutti
 */
#ifndef pamcalo_h
#define pamcalo_h

#include <TObject.h>

/**
 *   \brief PAMELA Calorimeter variables class
 *
 * This class contains observables obtained with data of the PAMELA
 * calorimeter.
 *
 */
class PamCalo : public TObject {

 public:
    //
    // constructors and destructors
    //
    PamCalo(); ///< default constructor (does nothing)
    void Clear(); ///< reset variables

    Float_t energy;  ///< Energy as measured by the tracking system
    Int_t pID;       ///< Particle ID, 0 = protons, 1 = electrons , 2 = positrons

    Float_t qtot;    ///< total energy detected (MIP)
    Float_t qmax;    ///< the maximum energy detected in a strip
    Float_t qtrack;  ///< the energy deposited in the strip closest to the track and the neighbouring strip on each side
    Float_t qcore;   ///< SUM(j=1,2)SUM(i=1,PLmax) Qhit(i,j)*i , where Qhit(i,j) is the energy released (MIP) in a cylinder of radius 2 Rm (Moli
ne is number 1 and the sum runs up to plane number PLmax, closest to the calculated electromagnetic shower maximum of the j-th view).
    Float_t qcyl;    ///< the measured energy deposited in a cylinder of radius 8 strips around the shower axis
    Float_t qlast;   ///< the same as "qcyl" but only for the last four planes and radius 4 strips.
    Float_t qpre;    ///< the same as "qcyl" but only for the first three planes
    Float_t qtr;     ///< the same as "qcyl" but with radius 4 strips
    Float_t q0;      ///< total energy release detected on plane 0
    Float_t q1;      ///< total energy release detected on plane 1
    Float_t q2;      ///< total energy release detected on plane 2
    Float_t q3;      ///< total energy release detected on plane 3
    Float_t q4;      ///< total energy release detected on plane 4
    Float_t q5;      ///< total energy release detected on plane 5

    Int_t nstrip;    ///< total number of strip hit
    Int_t ncore;     ///< SUM(j=1,2)SUM(i=1,PLmax) Nhit(i,j)*i , where Nhit(i,j) is the number of hits in a cylinder of radius 2 Rm (Moliere rad
umber 1 and the sum runs up to plane number PLmax, closest to the calculated electromagnetic shower maximum of the j-th view)
    Int_t noint;     ///< SUM(j=1,2)SUM(i=1,22) TH(i,j)*i , where TH(i,j) = 1 if the i-th plane of the j-th view has a cluster along (less than
(order of one MIP), otherwise TH(i,j) = 0
    Int_t ncyl;      ///< the number of strip hit in a cylinder of radius 8 strips around the shower axis
    Int_t nlast;     ///< the same as "ncyl" but only for the last four planes and radius 4 strips.
    Int_t npre;      ///< the same as "ncyl" but only for the first three planes
    Int_t ntr;       ///< the same as "ncyl" but with radius 4 strips

    //
    ClassDef(PamCalo,1);
};

#endif
```

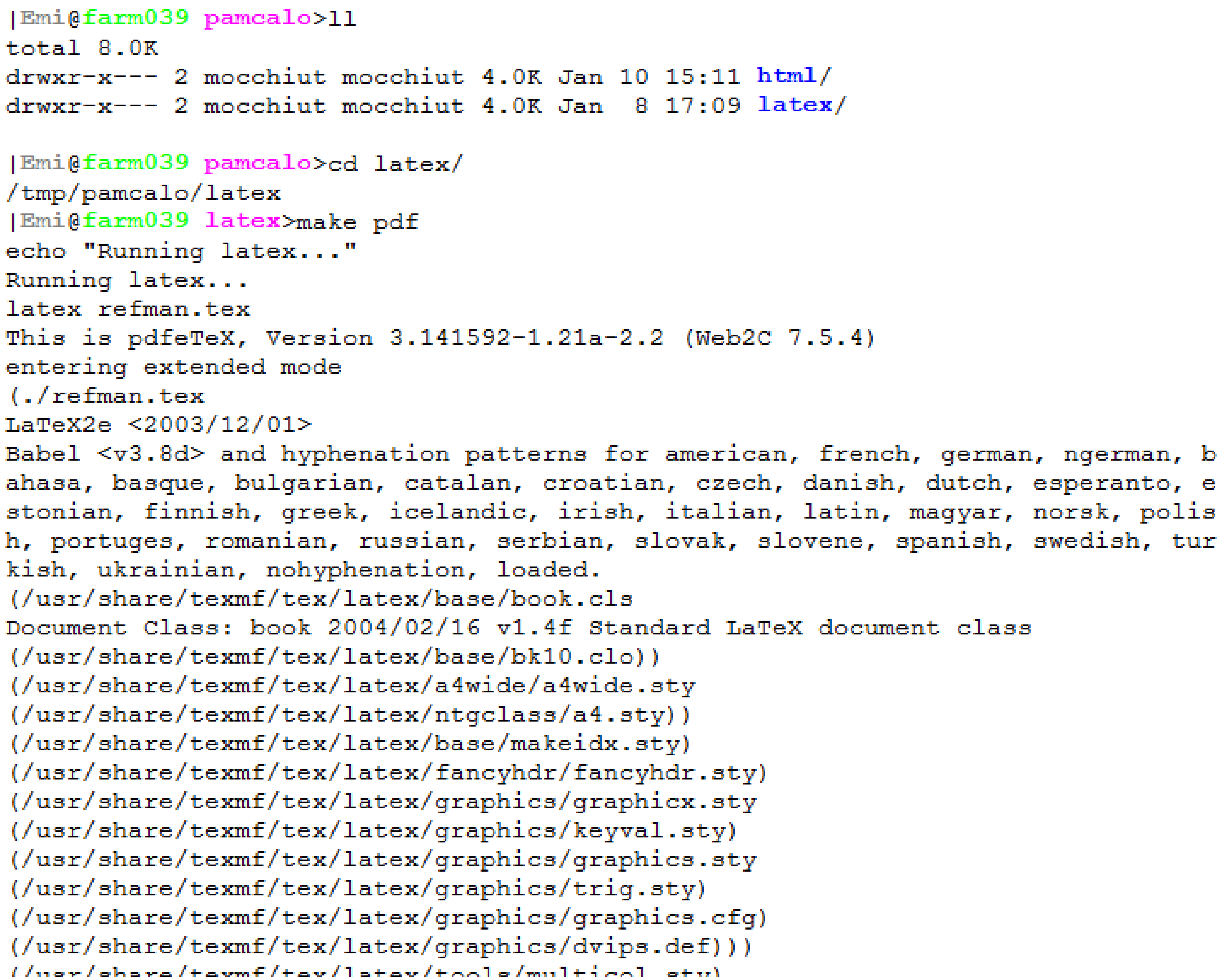
DOXYGEN commands!

Running doxygen



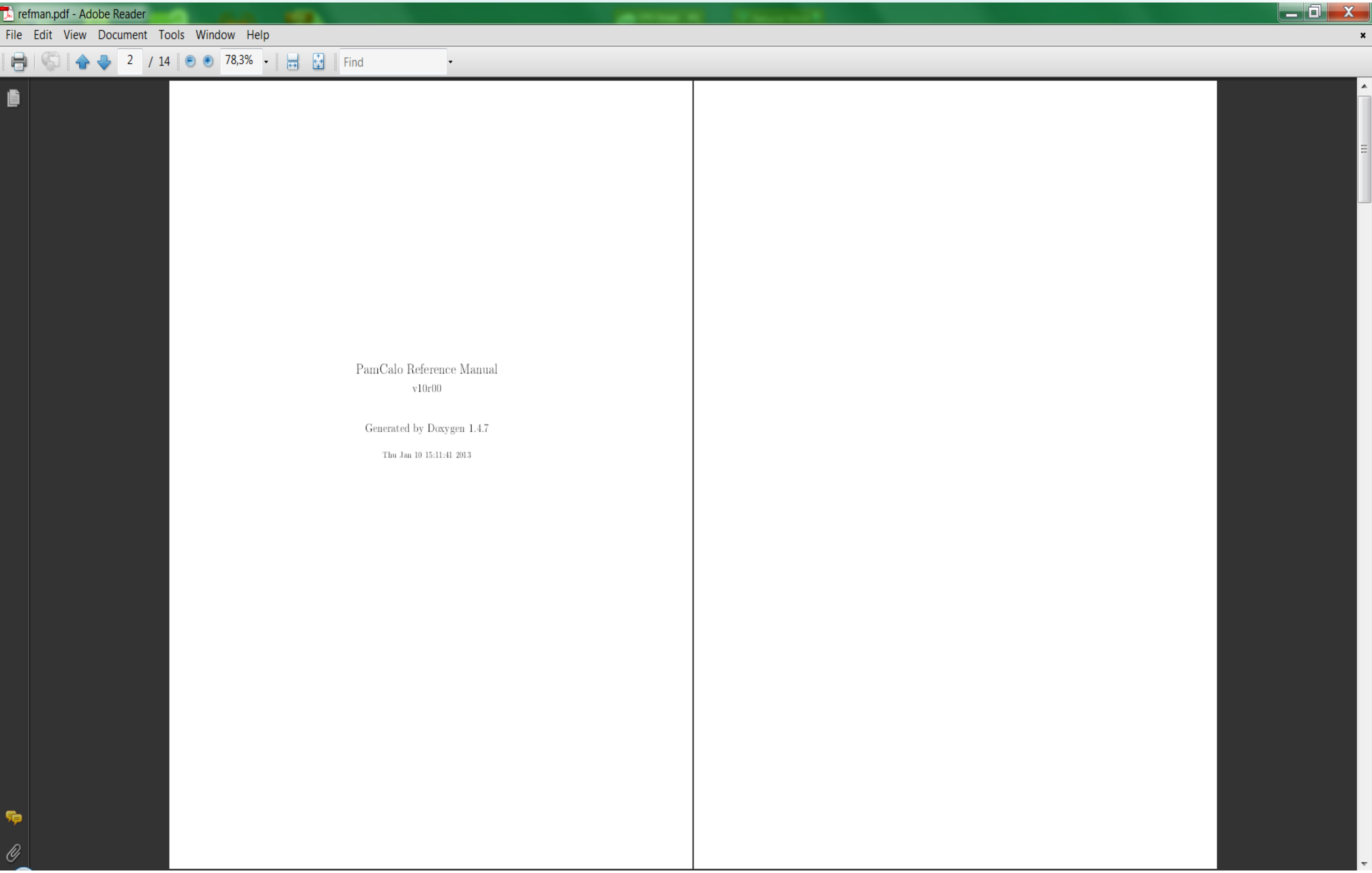
```
|Emi@farm039 PamCalo>ll
total 176K
drwxr-x--- 2 mocchiut mocchiut 16K Jan 10 15:10 doc/
drwxr-x--- 2 mocchiut mocchiut 16K Jan  8 17:10 lib/
drwxr-x--- 2 mocchiut mocchiut 16K Jan  8 17:10 bin/
drwxrwx--- 2 mocchiut wizard   16K Jan  8 17:10 src/
-rwxrwx--- 1 mocchiut wizard   13K Dec  5 09:24 Makefile*
drwxrwx--- 2 mocchiut wizard   16K Dec  5 09:23 inc/

|Emi@farm039 PamCalo>ls doc/
PamCalo.dox
|Emi@farm039 PamCalo>doxygen doc/PamCalo.dox
Searching for include files...
Searching for example files...
Searching for images...
Searching for dot files...
Searching for files to exclude
Searching for files to process...
Reading and parsing tag files
Preprocessing /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/doc/PamCalo.dox...
Parsing file /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/doc/PamCalo.dox...
Preprocessing /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/inc/PamCalo.h...
Parsing file /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/inc/PamCalo.h...
Preprocessing /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/inc/PamCaloLinkDef.h...
Parsing file /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/inc/PamCaloLinkDef.h...
Preprocessing /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/src/PamCalo.cpp...
Parsing file /gpfs/wizard/flight/src/offline/calo/flight/PamCalo/src/PamCalo.cpp...
Building group list...
Building directory list...
Building namespace list...
Building file list...
Searching for included using directives...
Building class list...
Associating documentation with classes...
Computing nesting relations for classes...
Searching for members imported via using declarations
```

Running doxygen



```
|Emi@farm039 pamcalo>ll
total 8.0K
drwxr-x--- 2 mocchiut mocchiut 4.0K Jan 10 15:11 html/
drwxr-x--- 2 mocchiut mocchiut 4.0K Jan  8 17:09 latex/

|Emi@farm039 pamcalo>cd latex/
/tmp/pamcalo/latex
|Emi@farm039 latex>make pdf
echo "Running latex..."
Running latex...
latex refman.tex
This is pdfeTeX, Version 3.141592-1.21a-2.2 (Web2C 7.5.4)
entering extended mode
(./refman.tex
LaTeX2e <2003/12/01>
Babel <v3.8d> and hyphenation patterns for american, french, german, ngerman, b
ahasa, basque, bulgarian, catalan, croatian, czech, danish, dutch, esperanto, e
stonian, finnish, greek, icelandic, irish, italian, latin, magyar, norsk, polis
h, portuges, romanian, russian, serbian, slovak, slovene, spanish, swedish, tur
kish, ukrainian, nohyphenation, loaded.
(/usr/share/texmf/tex/latex/base/book.cls
Document Class: book 2004/02/16 v1.4f Standard LaTeX document class
(/usr/share/texmf/tex/latex/base/bk10.clo))
(/usr/share/texmf/tex/latex/a4wide/a4wide.sty
(/usr/share/texmf/tex/latex/ntgclass/a4.sty))
(/usr/share/texmf/tex/latex/base/makeidx.sty)
(/usr/share/texmf/tex/latex/fancyhdr/fancyhdr.sty)
(/usr/share/texmf/tex/latex/graphics/graphicx.sty
(/usr/share/texmf/tex/latex/graphics/keyval.sty)
(/usr/share/texmf/tex/latex/graphics/graphics.sty
(/usr/share/texmf/tex/latex/graphics/trig.sty)
(/usr/share/texmf/tex/latex/graphics/graphics.cfg)
(/usr/share/texmf/tex/latex/graphics/dvips.def)))
(/usr/share/texmf/tex/latex/tools/multicol.sty)
```

Doxygen, output

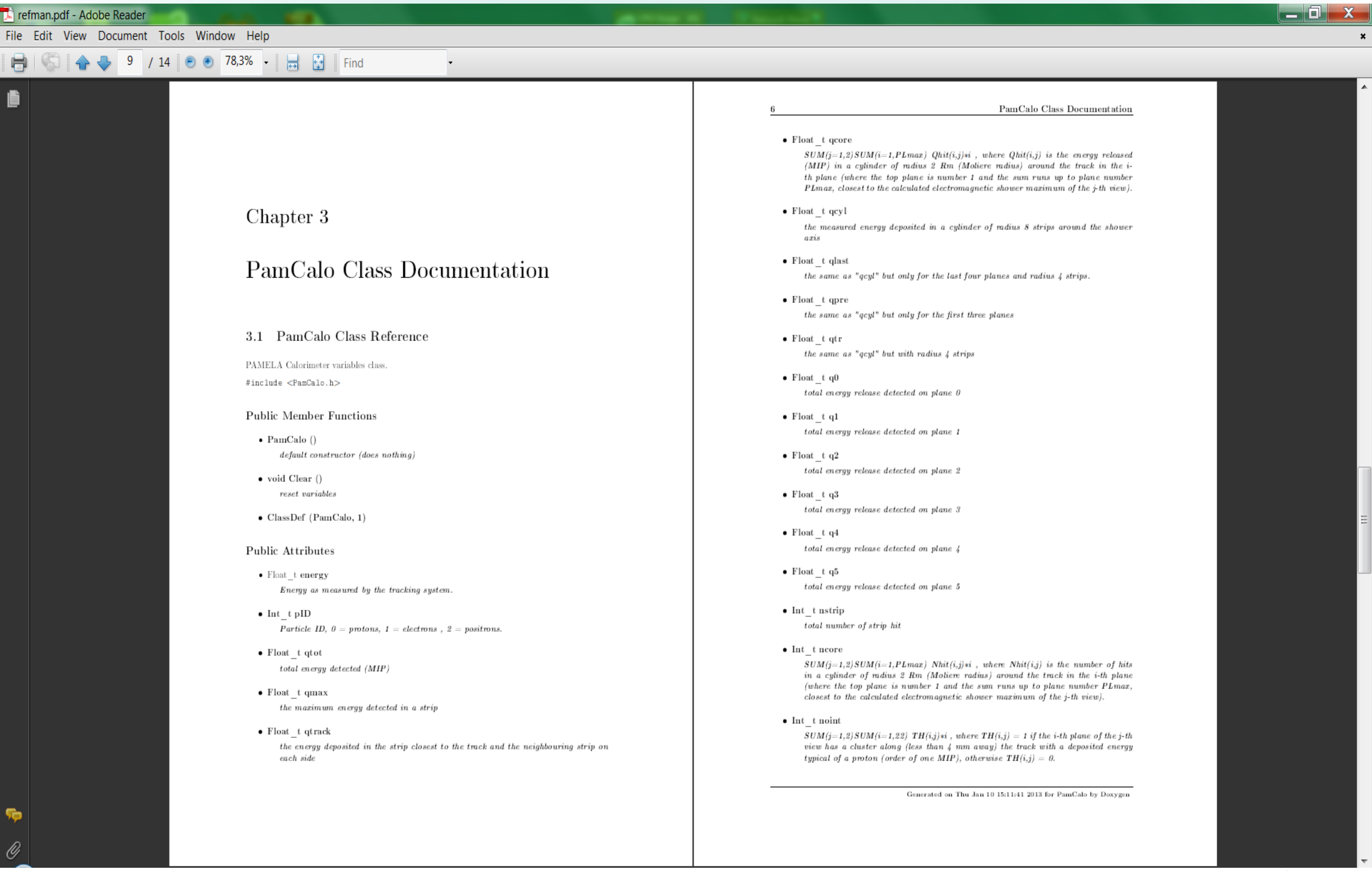

refman.pdf - Adobe Reader
File Edit View Document Tools Window Help
2 / 14
78,3%
Find

PamCalo Reference Manual
v10r00

Generated by Doxygen 1.4.7

Thu Jan 10 15:11:41 2013

Doxygen, output


refman.pdf - Adobe Reader
File Edit View Document Tools Window Help
9 / 14
78,3%
Find

6 PamCalo Class Documentation

Chapter 3

PamCalo Class Documentation

3.1 PamCalo Class Reference

PAMELA Calorimeter variables class.

`#include <PamCalo.h>`

Public Member Functions

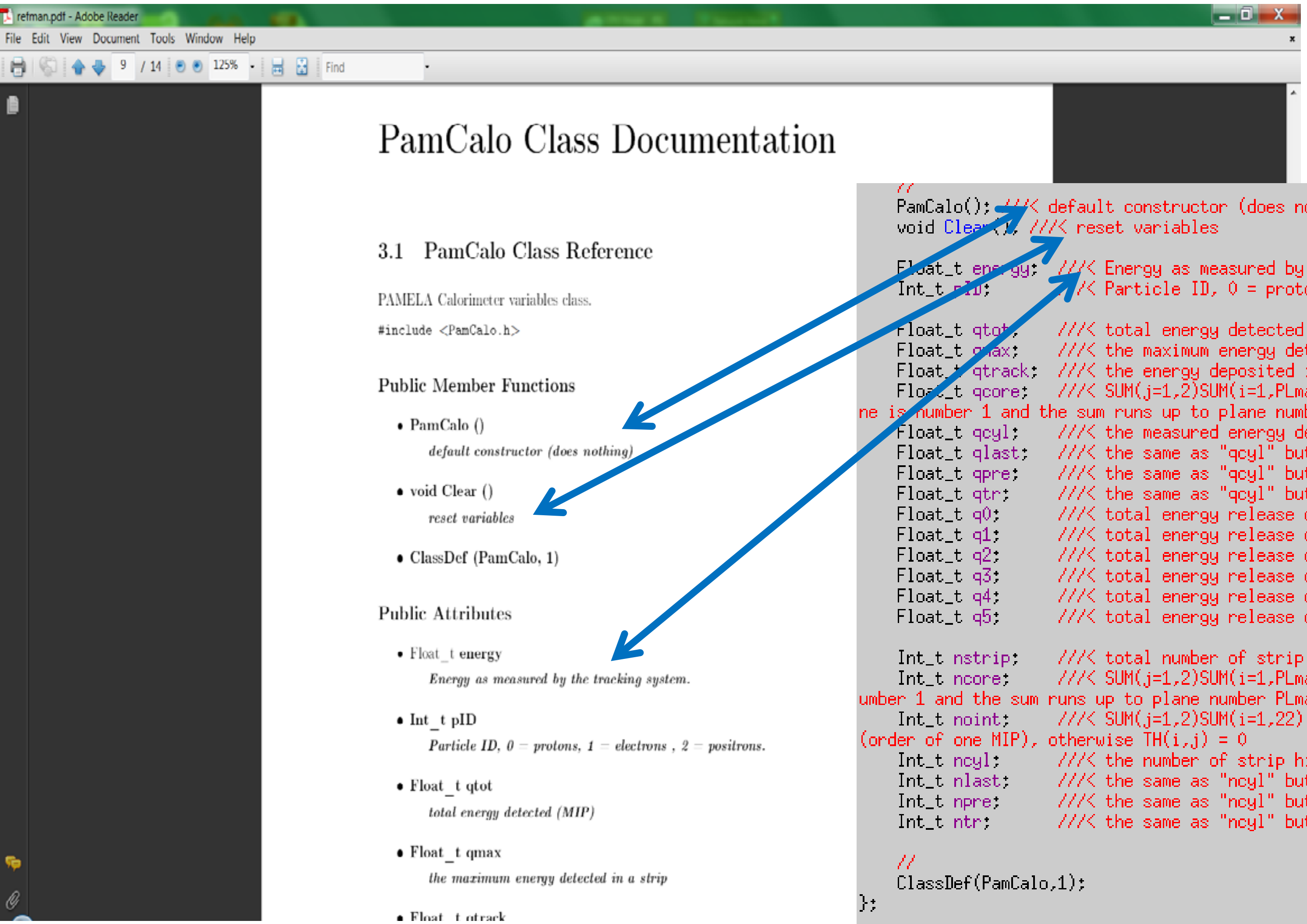
- PamCalo ()
  default constructor (does nothing)
- void Clear ()
  reset variables
- ClassDef (PamCalo, 1)

Public Attributes

- Float_t energy
  Energy as measured by the tracking system.
- Int_t pID
  Particle ID, 0 = protons, 1 = electrons , 2 = positrons.
- Float_t qtot
  total energy detected (MIP)
- Float_t qmax
  the maximum energy detected in a strip
- Float_t qtrack
  the energy deposited in the strip closest to the track and the neighbouring strip on each side
- Float_t qcore
  *SUM(j=1,2)SUM(i=1,PLmax) Qhit(i,j)*i , where Qhit(i,j) is the energy released (MIP) in a cylinder of radius 2 Rm (Moliere radius) around the track in the i-th plane (where the top plane is number 1 and the sum runs up to plane number PLmax, closest to the calculated electromagnetic shower maximum of the j-th view).*
- Float_t qcyl
  the measured energy deposited in a cylinder of radius 8 strips around the shower axis
- Float_t qlast
  the same as "qcyl" but only for the last four planes and radius 4 strips.
- Float_t qpre
  the same as "qcyl" but only for the first three planes
- Float_t qtr
  the same as "qcyl" but with radius 4 strips
- Float_t q0
  total energy release detected on plane 0
- Float_t q1
  total energy release detected on plane 1
- Float_t q2
  total energy release detected on plane 2
- Float_t q3
  total energy release detected on plane 3
- Float_t q4
  total energy release detected on plane 4
- Float_t q5
  total energy release detected on plane 5
- Int_t nstrip
  total number of strip hit
- Int_t ncore
  *SUM(j=1,2)SUM(i=1,PLmax) Nhit(i,j)*i , where Nhit(i,j) is the number of hits in a cylinder of radius 2 Rm (Moliere radius) around the track in the i-th plane (where the top plane is number 1 and the sum runs up to plane number PLmax, closest to the calculated electromagnetic shower maximum of the j-th view).*
- Int_t noint
  *SUM(j=1,2)SUM(i=1,22) TH(i,j)*i , where TH(i,j) = 1 if the i-th plane of the j-th view has a cluster along (less than 4 mm away) the track with a deposited energy typical of a proton (order of one MIP), otherwise TH(i,j) = 0.*

Generated on Thu Jan 10 15:11:41 2013 for PamCalo by Doxygen

Doxygen, output


refman.pdf - Adobe Reader
File Edit View Document Tools Window Help
9 / 14
125%
Find

PamCalo Class Documentation

3.1 PamCalo Class Reference

PAMELA Calorimeter variables class.

`#include <PamCalo.h>`

Public Member Functions

- PamCalo ()

  default constructor (does nothing)

- void Clear ()

  reset variables

- ClassDef (PamCalo, 1)

Public Attributes

- Float_t energy

  Energy as measured by the tracking system.

- Int_t pID

  Particle ID, 0 = protons, 1 = electrons , 2 = positrons.

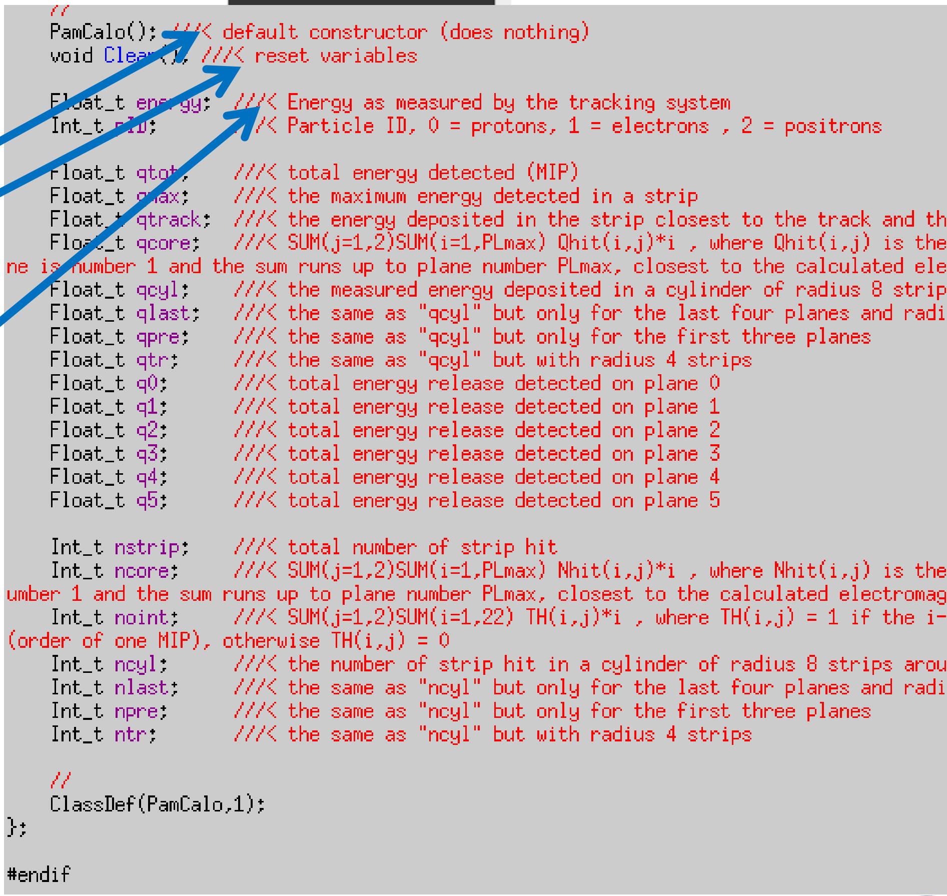
- Float_t qtot

  total energy detected (MIP)

- Float_t qmax

  the maximum energy detected in a strip

- Float_t qtrack



```
    //
    PamCalo(); ///< default constructor (does nothing)
    void Clear(); ///< reset variables

    Float_t energy;  ///< Energy as measured by the tracking system
    Int_t pID;       ///< Particle ID, 0 = protons, 1 = electrons , 2 = positrons

    Float_t qtot;    ///< total energy detected (MIP)
    Float_t qmax;    ///< the maximum energy detected in a strip
    Float_t qtrack;  ///< the energy deposited in the strip closest to the track and th
    Float_t qcore;   ///< SUM(j=1,2)SUM(i=1,PLmax) Qhit(i,j)*i , where Qhit(i,j) is the
ne is number 1 and the sum runs up to plane number PLmax, closest to the calculated ele
    Float_t qcyl;    ///< the measured energy deposited in a cylinder of radius 8 strip
    Float_t qlast;   ///< the same as "qcyl" but only for the last four planes and radi
    Float_t qpre;    ///< the same as "qcyl" but only for the first three planes
    Float_t qtr;     ///< the same as "qcyl" but with radius 4 strips
    Float_t q0;      ///< total energy release detected on plane 0
    Float_t q1;      ///< total energy release detected on plane 1
    Float_t q2;      ///< total energy release detected on plane 2
    Float_t q3;      ///< total energy release detected on plane 3
    Float_t q4;      ///< total energy release detected on plane 4
    Float_t q5;      ///< total energy release detected on plane 5

    Int_t nstrip;    ///< total number of strip hit
    Int_t ncore;     ///< SUM(j=1,2)SUM(i=1,PLmax) Nhit(i,j)*i , where Nhit(i,j) is the
umber 1 and the sum runs up to plane number PLmax, closest to the calculated electromag
    Int_t noint;     ///< SUM(j=1,2)SUM(i=1,22) TH(i,j)*i , where TH(i,j) = 1 if the i-
(order of one MIP), otherwise TH(i,j) = 0
    Int_t ncyl;      ///< the number of strip hit in a cylinder of radius 8 strips arou
    Int_t nlast;     ///< the same as "ncyl" but only for the last four planes and radi
    Int_t npre;      ///< the same as "ncyl" but only for the first three planes
    Int_t ntr;       ///< the same as "ncyl" but with radius 4 strips

    //
    ClassDef(PamCalo,1);
};

#endif
```

Repositories

Repositories, CVS and GIT

CVS and GIT (and many others! e.g. RCS, subversion, perforce, etc.) are version control system.

With version control system is meant the management of changes to documents, computer programs, large web sites, and other collections of information. Changes are usually identified by a number or letter code, termed the “revision number”, “revision level”, or simply “revision”.

For example, an initial set of files is “revision 1”. When the first change is made, the resulting set is “revision 2”, and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

The need for a logical way to organize and control revisions has existed for almost as long as writing has existed, but revision control became much more important, and complicated, when the era of computing began. The numbering of book editions and of specification revisions are examples that date back to the print-only era. Today, the most capable (as well as complex) revision control systems are those used in software development, where a team of people may change the same files.

Repositories, CVS and GIT

Revisions can be compared, restored, and with some types of files, merged.

The need for a logical way to organize and control revisions has existed for almost as long as writing has existed, but revision control became much more important, and complicated, when the era of computing began. The numbering of book editions and of specification revisions are examples that date back to the print-only era.

Today, the most capable (as well as complex) revision control systems are those used in software development, where a team of people may change the same files.

Repositories, CVS and GIT

CVS: http://cvs.nongnu.org/

CVS was one of the most used version control system till some years ago.

GIT: http://git-scm.com/

GIT is very actual and interesting, you can try it from the web browser by clicking here: http://try.github.com

2017/03/24 – take home messages

- Avoid crash!
- **READ WARNING AND ERROR MESSAGES**
- CMake easy and comfty... but you need to be clean and make order in your dirs