



# Laboratorio di Elettromagnetismo e Ottica– C++/ROOT

## Lezione V

Silvia Arcelli

# Interpretation vs Compilation

---

Due modi per lavorare in modalità **compilata** (più veloce e maggiormente controllata dal punto di vista della sintassi). In entrambi i casi, rispetto alla modalità interprete, occorre inserire nella macro tutti gli header necessari.

- **Compilare la Macro da linea di comando di ROOT**
  - `root [0] .L myMacro.C+ (or with ++, forza la ricompilazione)`
  - Oppure, `root [0] gROOT->LoadMacro("myMacro.C+")`
  - `root [1] myMacro() //per mandare in esecuzione`
- **Compilare la Macro esternamente, da SHELL**
  - `>g++ -o ExampleMacro .exe myMacro.C `root-config --cflags --libs``
  - `> ./ExampleMacro.exe`
  - **In questo caso bisogna aggiungere il main()!**

# ESERCIZIO.V1.4

---

- Modifichiamo la macro della scorsa lezione per poterla eseguire in modalità compilata, utilizzando i metodi illustrati nella slide precedente

**Macro myMacroFinal.C**

# Esempio di utilizzo di TList

---

- TList: **classe contenitore**, sul modello della list della stl
- Nella TList potete anche mettere oggetti eterogenei, a patto che derivino da **TObject** (classe «madre» di tutti gli oggetti di ROOT):

Per creare una Tlist:

```
TList * list= new TList(); //constructing the Tlist
```

Per aggiungere un oggetto obj (attraverso il puntatore) a una Tlist:

```
list->Add(obj); // obj is a pointer to any object  
                  deriving from TObject
```

Per accedere all'i-esimo elemento di una Tlist:

```
list->At(i); //returns the i-th element in the list
```

# Esempio di utilizzo di TList

---

- Definire tre istogrammi, 2 **unidimensionali** e 1 **bidimensionale**, e un grafico (TGraphErrors).
- Riempire gli istogrammi con distribuzioni gaussiane 1D e 2-D usando il metodo FillRandom, riempire il grafico con arrays nativi.
- inserirli in una TList
- inserire anche il grafico nella TList e disegnare il tutto accedendo a quanto inserito nella TList attraverso i suoi metodi (macro testList.C su Virtuale)

**Macro testList.C**

# Esempio di utilizzo di TList

---

//Filling The TList

```
TList * testList=new TList();
```

```
for(Int_t i=0;i<3;i++) testList->Add(h[i]);
```

```
testList->Add(graph);
```


// per stampare a schermo il contenuto della  
TList

```
testList->Print();
```

# Esempio di utilizzo di TList

//Drawing objects contained in the TList

```
TCanvas *c1 = new TCanvas("c1","TList example",200,10,600,400);
c1->Divide(2,2);
for(Int_t i=0;i<4;i++){
    c1->cd(i+1);
    if(!testList->At(i)->InheritsFrom("TGraph")){
        if(!testList->At(i)->InheritsFrom("TH2")){
            testList->At(i)->Draw("H");
            testList->At(i)->Draw("E,P,SAME");
        } else testList->At(i)->Draw("SURF1");
    }
    else testList->At(i)->Draw("APE");
}
```



Esempio di funzionalità di ROOT di **Run Time Type Introspection (RTTI)**

# Esempio di utilizzo di Tlist, V2

---

- Scrivere la TList su un root File
- «Homework» per voi: Adattate la macro per essere eseguita in modalità compilata

**Macro testList.C**



# ROOT File

ROOT ha un'interfaccia di I/O propria, implementata nella classe **TFile**:

Costruttore di TFile:

```
TFile (const char *fname, Option_t *option="")
```

Per aprire un file di nome example.root:

```
TFile *file = new TFile("example.root")
```

**In lettura**

```
TFile *file = new TFile("example.root", "RECREATE")
```

**In scrittura**

```
TFile *file = new TFile("example.root", "UPDATE")
```

**in modifica**

Option	Description
NEW or CREATE	Create a new file and open it for writing, if the file already exists the file is not opened.
RECREATE	Create a new file, if the file already exists it will be overwritten.
UPDATE	Open an existing file for writing. If no file exists, it is created.
READ	Open an existing file for reading (default).

# ROOT File

Per scrivere su un file root tutti gli istogrammi in memoria nella sessione corrente si usa il metodo :

**Int\_t TFile::Write ()**

Per scrivere su un file root un **singolo oggetto** si usa il metodo :

**TObject::Write ()**

Per chiudere il file root si usa il metodo :

**Int\_t TFile::Close()**

## Esempio :

```
TFile *file=new TFile("example.root","RECREATE");
TH1F *h1 = new TH1F("h1","title1",100,0.,1);
TH1F *h2 = new TH1F("h2","title2",100,0.,1);
file->Write();
h2->Write();
file->Close();
```

**Create the root file**

**Create 2 histos**

**Write both on file**

**Write h2 on file**

**Close file**

10

# Accesso al contenuto dei file di ROOT

---

- Per poi accedere all'istogramma da root, da linea di comando o in una macro:

```
root[] TFile *file =new TFile("example.root"); apertura del file  
                                              (read mode)  
root[] file->ls(); listing del contenuto del file  
  
root[] TH1F *histo=(TH1F*)file->Get("histo"); estraggo l'istogramma  
                                              attraverso il suo nome
```

- Attraverso comandi C++ diretti o attraverso macro o interfaccia grafica potete ulteriormente manipolare il vostro oggetto (proprietà, operazioni, grafica)

# Decouple Object from TFile

---

- Per evitare che gli istogrammi “spariscano” dalla canvas quando si chiude il file root si possono usare i tre seguenti metodi (uno vale l'altro):
  - `TH1::AddDirectory(kFALSE);` (statico)
  - `TH1::SetDirectory(0);` (non statico)
  - `TH1::DrawCopy();` (non statico)

# How to include a Class in ROOT

---

- Come utilizzare una classe definita da voi, che eventualmente fa uso a sua volta anche di classi di ROOT, all'interno di ROOT
- Scriviamo una classe di esempio:
  - **MyClass.h** (intestazione)
  - **MyClass.cxx** (implementazione)
- Seguiamo per quanto possibile le coding conventions

# Esempio

---

- definiamo un unico membro dato (**privato**) che è un puntatore a un istogramma unidimensionale (TH1F\*).

E poi, i metodi (**pubblici**):

- **costruttore di default e costruttore parametrico** (argomento un puntatore TH1F\*), e il **distruttore**.
- un metodo **Generate** (argomenti: un puntatore a funzione ROOT, TF1\*, un intero nGen) per riempire l'istogramma con la distribuzione definita dalla funzione TF1, utilizzando il metodo FillRandom(...). Generate un milione di occorrenze **con una p.d.f.  $f(x)=x$  in  $[0,1]$** .
- un metodo che disegni l'istogramma, **ShowHisto()**
- un getter per l'attributo interno, **GetHisto()**

# MyClass.h

---

```
#ifndef MYCLASS_H
#define MYCLASS_H

#include "TF1.h"
#include "TH1F.h"
class MyClass{
public:
MyClass(); //def ctor
MyClass(TH1F *hIn); //parametric ctor
//public methods
void Generate(TF1*f, int nGen); //generate according to a given function
void ShowHisto() const; //display the histogram
TH1F* GetHisto() const; //retrieve the histogram
~MyClass(); //dtor
private:
TH1F * h_; //pointer to a histogram
};
#endif
```

# MyClass.cxx

---

```
#include "MyClass.h"

MyClass::MyClass() {h_ = new TH1F("h", "titolo", 100, -5, 5);} //def ctor
MyClass::MyClass(TH1F *hIn) {h_ = new TH1F(*hIn);} //parametric ctor

void MyClass::Generate(TF1*f, int nGen) {
    h_>GetXaxis()>SetRangeUser(f->GetXmin(), f->GetXmax()); //set the
range to f range
    h_>FillRandom(f->GetName(), nGen); //accepts char* as arg, must
use GetName()
}

void MyClass::ShowHisto() const { h_>Draw();}
TH1F* MyClass::GetHisto() const {return h_;}
MyClass::~MyClass() {delete h_;} //dtor
```



# How to include a Class in ROOT

---

- **Compiliamo** da linea di comando la classe:
  - `gROOT->LoadMacro ("MyClass.cxx+")`
- Nella Macro `myClassMacro.C`, costruiamo l'oggetto di tipo `MyClass` secondo quanto richiesto dall'esercizio e invochiamo i metodi pubblici che ci consentono di:
  - riempire l'istogramma con la funzione richiesta (metodo `Generate`)
  - disegnarlo sulla canvas di default (metodo `ShowHisto`).
- Se la libreria `MyClass_cxx.so` è già presente, per evitare di fare ogni volta `LoadMacro` si può inserire all'inizio:  
`R__LOAD_LIBRARY(MyClass_cxx.so)`

# myClassMacro.C

---

Macro myClassMacro.C, funzione CreateAndWriteMyClass()

```
void CreateAndWriteMyClass() {
    TF1 *f = new TF1("f","x",0.,1.); //define the input fuction
    TH1F *hIn = new TH1F("hIn","input histo",100,0.,1.); //define
the input histo to the ctor
    MyClass * A = new MyClass(hIn); //create object A
    A->Generate(f,1E6); //Fill the histo according p.d.f. «f»
    A->ShowHisto(); //Draw the
histo
    return;
}
```

# Writing your own Object to a ROOT file

---

Un oggetto di una classe da noi definita lo possiamo anche **scrivere su un file ROOT**. Come si procede? Tre semplici operazioni:

Modifichiamo MyClass:

- inserendo nel **file di intestazione MyClass.h**:
  1. **l'ereditarietà pubblica da TObject**
  2. la chiamata alla macro **ClassDef(ClassName,N)**
- Inserendo nel file **di implementazione MyClass.cxx**:
  3. la chiamata a **ClassImp(ClassName)**

# MyClass.h, nuova versione

```
#ifndef MYCLASS_H
#define MYCLASS_H
```

includere statement in rosso!

```
#include "TF1.h"
#include "TH1F.h"
class MyClass: public TObject{ //public inheritance from TObject
public:
    MyClass(); //def ctor
    MyClass(TH1F *hIn); //parametric ctor
    //public methods
    void Generate(TF1*f, int nGen); //generate according to a given function
    void ShowHisto() const; //display the histogram
    TH1F* GetHisto() const; //retrieve the histogram
    ~MyClass(); //dtor
private:
    TH1F * h_; //pointer to a histogram
    //for persistency, to make MyClass Objects writable on root files
    ClassDef(MyClass,1)
};
#endif
```

# MyClass.cxx, nuova versione

includere statement in rosso!

```
#include "MyClass.h"

MyClass::MyClass() {h_ = new TH1F("h", "titolo", 100, -5, 5);} //def Ctor
MyClass::MyClass(TH1F *hIn) {h_ = new TH1F(*hIn);} //parametric ctor

void MyClass::Generate(TF1*f, int nGen) {
    h_>GetXaxis()->SetRangeUser(f->GetXmin(), f->GetXmax()); //set the range to f range
    h_>FillRandom(f->GetName(), nGen); //accepts char* as arg, must use GetName()
}

void MyClass::ShowHisto() const { h_>Draw();}
TH1F* MyClass::GetHisto() const {return h_;}
MyClass::~MyClass() {delete h_;} //dtor
//for persistency, to make MyClass Objects writable on root files
ClassImp(MyClass)
```

# Writing your own Object to a ROOT file

---

- Proviamo a scrivere l'oggetto su un file ROOT. Ora MyClass eredita da TObject, oggetto base di tutti gli oggetti di ROOT, e come gli altri è scrivibile su un file ROOT.
- Modifichiamo CreateAndWriteMyClass() aggiungendo:
  - Apertura file ROOT
  - Scrittura dell'oggetto A
  - Chiusura del file ROOT

# myClassMacro.C

Macro myClassMacro.C, CreateAndWriteMyClass()

```
void CreateAndWriteMyClass() {
  TF1 *f = new TF1("f","x",0.,1.); //define the input fuction
  TH1F *hIn = new TH1F("hIn","input histo",100,0.,1.); //define the
  input histo to the ctor
  MyClass * A = new MyClass(hIn); //create A
  A->Generate(f,10000); //Fill the histo
  A->ShowHisto(); //Draw the histo

  //Writing the user-defined object to a root file

  TFile *file = new TFile("test.root","recreate"); //open the file
  A->Write("A"); //write the object to a root file
  file->Close(); //close the file
  return;
}
```

# Reading your own Object from a ROOT file

---

Per recuperare l'oggetto, usate il metodo `TFile::Get("name")`, come per un oggetto nativo di ROOT. Avete un sistema di **persistenza per gli oggetti**.

Macro `myClassMacro.C`, **funzione `ReadMyClass()`**

```
void ReadMyClass() {  
  TFile *file = new TFile("test.root");  
  MyClass *B = (MyClass*)file->Get("A");  
  B->ShowHisto();  
  return;  
}
```

PS: è consigliabile utilizzare sempre un «cast» esplicito di tipo premettendolo al metodo `Get(...)`, ovvero: `MyClass *B= (MyClass *) file->Get("A") ;`