

GUIDA ESAME SCRITTO ROOT

Riccardo Fabbri

Nicola Mercuriali

Maggio 2021

1 CREAZIONE E GESTIONE OGGETTI

- CLASSE *NOME = new CLASSE(opt) [es: `TH1F *histo = new TH1F(...)`]
- NOME -> METODO(opt) [es: `histo -> Fill(...)`]

2 ISTOGRAMMI

`TH1F("nome", "titolo", nBin, xmin, xmax)` Crea un istogramma 1D di n bin sul dominio [xmin, xmax]

- `Fill(x)` o `Fill(x, nOccorrenze)` Riempie l'istogramma con il valore x per il numero di occorrenze (se indicate)
- `Draw("opt")` Disegna l'istogramma
 - "SAME" Disegna sul grafico precedente
 - "HISTO" Usa una linea continua
 - "E" Disegna le barre di errore
 - "P" Disegna solo i punti
- `gStyle -> SetOptStat(ourmen)` Setta cosa visualizzare nella legenda dell'istogramma (1= on, 0 = off)
[es: `SetOptStat(2210)` N. di entrate, media con errore e dev. std. con errore]
 - o = Numero di OVERFLOW
 - u = Numero di UNDERFLOW
 - r = Deviazione standard (r = 2 → Deviazione standard + errore)
 - m = Media (m = 2 → Media + errore)
 - e = Numero di entrate
 - n = Nome dell'istogramma

Metodi statistici:

- `GetMean()` Restituisce la media del campione di dati
 - `GetMeanError()` Restituisce l'errore sulla media
- `GetRMS()` Restituisce la deviazione standard
 - `GetRMSError()` Restituisce l'errore sulla deviazione standard
- `GetMaximum()` / `GetMinimum()` Restituisce il massimo/minimo dell'istogramma

- `GetEntries()` Restituisce il numero di ingressi inseriti
- `Integral()` Come `GetEntries`, ma non tiene conto dei pesi assegnati
- `Integral(bin1, bin2)` Restituisce l'integrale tra i due bin

Metodi dei bin:

- `GetMaximumBin()` Restituisce l'indice del bin contenente il valore massimo
- `GetBinCenter(iBin)` Restituisce il valore centrale dell'intervallo del bin i-esimo (parte da 1)
- `GetBinContent(iBin)` Restituisce il numero di elementi del bin i-esimo (parte da 1)
 - `GetBinError(iBin)` Restituisce l'errore sul bin i-esimo (parte da 1)
- `GetBinContent(0)` Restituisce il numero di UNDERFLOW
- `GetBinContent(nBins+1)` Restituisce il numero di OVERFLOW
- `SetBinContent(iBin, val)` Assegna al bin i-esimo il valore *val*

Metodi estetici:

- `SetMarkerStyle(#)` Modifica il tipo di marker (5 = \times , 4 = \circ , 8 = \bullet)
 - `SetMarkerSize(dim)` Modifica la dimensione dei marker
- `SetLineStyle(#)` Modifica il tipo di linea (5 = tratteggiata stretta, 9 = tratteggiata larga)
 - `SetLineWidth(dim)` Modifica lo spessore della linea
 - `SetLineColor(#)` Modifica il colore della linea (1, 2, 3, 4, 5)
- `SetFillColor(#)` Modifica il colore dell'area sottesa dal grafico (1, 2, 3, 4, 5)
- `SetTitle("Titolo"; "asse X"; "asse Y")` Inserisce il titolo e i nomi degli assi
 - `SetTitleSize()` Modifica la dimensione del titolo
 - `SetTitleOffset()` Modifica l'offset del titolo
- `GetXaxis()/GetYaxis() -> SetTitleSize()/SetTitleOffset()` Modifica dim/offset del nome dell'asse X/Y

Operazioni (posti h1, h2, h3 istogrammi):

- `h1 -> Sumw2()` Da usare per un corretto calcolo degli errori nell'istogramma derivato
- `h3 -> c*h1` Esegue la moltiplicazione ($c1 \cdot h1$) e la assegna ad h3
- `h3 -> Divide(h1, h2, c1, c2)` Esegue la divisione $[(c1 \cdot h1)/(c2 \cdot h2)]$ e la assegna ad h3
- `h3 -> Add(h1, h2, c1, c2)` Esegue la somma $[(c1 \cdot h1) + (c2 \cdot h2)]$ e la assegna ad h3

3 GRAFICI

`TGraph(n, \vec{x} , \vec{y})` Crea un grafico 2D di n punti usando come coordinate gli \vec{x} e \vec{y}

- `TGraphErrors(n, \vec{x} , \vec{y} , $\vec{e_x}$, $\vec{e_y}$)` Crea il grafico con anche le barre di errore usando gli array $\vec{e_x}$, $\vec{e_y}$

`TGraph("nomeFile", "%lg %lg")` Crea un grafico 2D da un file contenete almeno due colonne di dati (per saltarne si aggiunge `"%*lg"` per ogni colonna non necessaria)

- `TGraphErrors("nomeFile", "%lg %lg %lg %lg")` Crea il grafico con anche le barre di errore usando la terza e quarta colonna
- `AddPoint(x, y)` Inserisce il punto (x,y) in coda al grafico
- `SetPoint(i, x, y)` Assegna al punto i-esimo le coordinate (x, y)
- `GetPoint(i, x, y)` Assegna alle variabili (x,y) le coordinate del punto i-esimo
- `GetX()` / `GetY()` Restituisce l'array \vec{x}/\vec{y}
- `GetN()` Restituisce il numero di punti
- `Draw("opt")` Disegna il grafico
 - "SAME" Disegna sul grafico precedente
 - "A" Disegna gli assi
 - "P" Disegna solo i punti
 - "E" Disegna le barre di errore (solo per `TGraphErrors`)

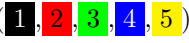
Metodi statistici:

- `Integral()` Restituisce l'area sottesa dal grafico
- `GetCorrelationFactor()` Restituisce il fattore di correlazione del grafico
- `GetCovariance()` Restituisce la covarianza del grafico

Metodi estetici: *come per ISTOGRAMMI*

4 LEGENDA

`TLegend(.1, .7, .3, .9, "nome")` Crea una legenda vuota da abbinare a un grafico (i numeri indicano la dimensione e la posizione della legenda, vanno intesi come [x1, y1, x2, y2])

- `AddEntry(NOME, "descrizione")` Inserisce un oggetto nella legenda e gli associa una descrizione
- `SetFillColor(#)` Modifica il colore della legenda ()
- `Draw("SAME")` Stampa la legenda sul grafico (se stampato in precedenza)

5 FUNZIONI

`TF1("nome", "funzione", xmin, xmax)` Crea una funzione ad una variabile sul dominio $[xmin, xmax]$, eventuali parametri vanno indicati come `[i]`

(Per funzioni definite a tratti: es: $f(x) * (x \geq a \ \&\& \ x < b) + g(x) * (x \geq b \ \&\& \ x < c) + \dots$)

- `Draw("opt")` Disegna la funzione ("SAME" Disegna sul grafico precedente)
 - `DrawDerivative()` Disegna la funzione derivata prima
 - `DrawIntegral()` Disegna la funzione integrale
- `SetParameter(i, val)` Assegna al parametro i-esimo il valore *val*
 - `SetParameters(val1, val2, ..., valN)` Assegna al parametro i-esimo il valore i-esimo (i da 1 a N)
 - `SetParLimits(i, xmin, xmax)` Definisce il range $[xmin, xmax]$ del parametro i-esimo
- `Eval(x)` Restituisce in valore di $f(x)$
- `Derivative(x)` / `Derivative2(x)` / `Derivative3(x)` Restituisce la derivata prima/seconda/terza nel punto x
- `Integral(a, b)` Restituisce l'integrale nell'intervallo $[a, b]$

6 FITTING

`NOME -> Fit("funzione", "opt")` È un metodo che fitta la funzione sull'oggetto a cui è applicato (Funzioni predefinite: "gaus", "expo", "pol1", "pol2", "pol3")

- "R" Usa il range della funzione per il fit (di default usa il range dell'oggetto)
- "L" Usa il metodo di massima verosimiglianza per il fit (di default usa il metodo del chiquadro)
- "Q" Non stampa nessuna informazione del fit (di default ne stampa un po')
- "V" Stampa tutte le informazioni del fit (di default ne stampa un po')
- `gStyle -> SetOptFit(pcev)` Setta quali informazioni del fit visualizzare sul grafico (1= on, 0 = off)
 - p = Probabilità
 - c = Chiquadro / Numero di gradi di libertà
 - e = Errori sui parametri
 - v = Parametri

Metodi del fit (da applicare alla funzione):

- `GetFunction()` Restituisce la funzione utilizzata nel fit (da applicare all'istogramma)
- `GetParameter(i)` Restituisce il valore del parametro i-esimo
 - `GetParError(i)` Restituisce l'errore sul parametro i-esimo
- `GetParameters(par)` Inserisce nell'array *par* i valori dei parametri
 - `GetParErrors(epar)` Inserisce nell'array *epar* gli errori sui parametri
- `GetChisquare()` Restituisce il valore del chiquadro
- `GetNDF()` Restituisce il numero di gradi di libertà

7 RANDOM

`gRandom -> SetSeed(seed)` Setta il seed per le generazioni random, importante per avere risultati consistenti

- `f1 -> GetRandom()` Genera un valore casuale dalla funzione `f1` definita in precedenza
- `h1 -> FillRandom("f1", n)` Filla l'istogramma `h1` con `n` estrazioni casuali dalla funzione `f1`
- `gRandom -> PDF` Fa un'estrazione casuale sulla base della PDF specificata come metodo
 - `Rndm()` Uniforme in $[0, 1]$
 - `Uniform(a, b)` Uniforme in $[a, b]$
 - `Gaus(μ , σ)` Gaussiana di media μ e deviazione σ
 - `Poisson(μ)` Poissoniana di media μ
 - `Binomial(n , p)` Binomiale di n estrazioni con probabilità di successo p
 - `Exp(τ)` Esponenziale decrescente

8 CANVAS

`TCanvas("nome", "titolo", larghezza, altezza)` Crea una Canvas vuota, richiede che l'xServer sia attivo

- `Divide(c , r)` Divide la canvas in una "griglia" di c colonne e r righe, i riquadri sono numerati dall'alto a sinistra partendo da 1
- `cd(i)` Seleziona il riquadro i -esimo della Canvas
- `Print("nomeFile")` Stampa la Canvas sul file

9 FILE

`TFile("nomeFile", "opt")` Crea un oggetto FILE associato a un file nella directory sulla base delle opzioni

- "NEW" o "CREATE" Crea un nuovo file e lo apre, ma se il file già esiste non lo apre
- "RECREATE" Crea un nuovo file e lo apre, se il file già esiste lo sovrascrive
- "UPDATE" Apre un file esistente, se il file non esiste lo crea
- "READ" Apre un file esistente in lettura (opzione di default)

Metodi dei file:

- `NOME -> Write()` Scrive l'oggetto a cui è applicato sul file aperto
- `Get("nome")` Restituisce l'oggetto indicato dal nome, da abbinare a uno static cast
- `Close()` Chiude il file aperto, importante per evitare problemi

10 BENCHMARK

gBenchmark -> METODO

- `Start("nome")` Avvia il benchmark e gli assegna il nome
- `Stop("nome")` Interrompe il benchmark con il nome corrispondente
- `Print("nome")` Stampa i dati del benchmark con il nome corrispondente
- `Show("nome")` Interrompe e stampa i dati del benchmark con il nome corrispondente (Stop+Print)

11 ESEMPI

11.1 Creare una macro e runnarla

```
/*   nel file macro.C   */

void macro() {
    ...                //comandi della macro
}

/*   sulla console di root   */

root [0] .L macro.C //carica il file, da fare ogni volta che lo si modifica
root [1] macro()    //esegue la macro, ovvero i comandi in "macro()"
```

11.2 Fillare un istogramma con un file

```
TH1F *histo = new TH1F("histo", "maxwell", 100, 0, 15);    //crea l'istogramma

ifstream in;
in.open("maxwell.dat");    //nome del file
while (in.good()) {
    Float_t x, y;          //uso "Float_t" al posto di "float" per motivi di compatibilità
    in >> x >> y;
    histo->Fill(y);        //filla l'istogramma
}
in.close();
```

11.3 Stampare dati sulla console

```
/*   ""\n"" o "endl" terminano la riga   */

cout << "N. ingressi: " << histo->GetEntries() << "\n";
cout << "Media: " << histo->GetMean() << " +/- " << histo->GetMeanError() << endl;
cout << "Dev. st.: " << histo->GetRMS() << " +/- " << histo->GetRMSError() << endl;
```

11.4 Fittare un grafico o un istogramma

```
/* per un istogramma è uguale */

const Int_t n = 10; //
double x_val[n] = {1,2,3,4,5,6,7,8,9,10}; //setta numero di punti e coordinate
double y_val[n] = {6,12,14,20,22,24,35,45,44,53}; //

TGraph *graf = new TGraph(n, x_val, y_val); //crea il grafico

TF1 *fit = new TF1("f", "x*[0] + [1]", 0, 11); //crea la funzione di fit (lineare)
graf->Fit(fit, "Q"); //fitta la funzione senza stampare nulla

TCanvas *c = new TCanvas(); //
graf->Draw("A, P"); //stampa il grafico e il fit
fit->Draw("SAME"); //
```

11.5 Legenda di un grafico

```
TLegend *leg = new TLegend(.1, .7, .3, .9, "Legenda"); //crea la legenda
leg->AddEntry(graf, "Grafico"); //aggiunge il grafico alla legenda
leg->AddEntry(fit, "Fit"); //aggiunge anche il fit
leg->Draw("SAME");
```

11.6 Fillare un istogramma con estrazioni random

```
gRandom->SetSeed(420); //setta il seed per le estrazioni
Int_t nEst = 1000; //setta il numero di estrazioni

TH1F *histo = new TH1F("histo", "gauss", 100, -10, 10); //crea l'istogramma

for (int i=0; i<nEst; ++i) { //per ogni estrazione
    Float_t x = gRandom->Gaus(0, 1); //prende la variabile x da una PDF gaussiana
    histo->Fill(x); //e la inserisce nell'istogramma
}
```

11.7 Dividere la canvas

```
TCanvas *c = new TCanvas();
c->Divide(2,1); //divide la canvas in 2 riquadri affiancati

c->cd(1); //seleziona il riquadro di sinistra
histo->Draw("HISTO"); //stampa l'istogramma

c->cd(2); //seleziona il riquadro di destra
graf->Draw("A, P"); //
fit->Draw("SAME"); // stampa il grafico con il fit e la legenda
leg->Draw("SAME"); //
```

11.8 Salvare la canvas

```
TCanvas *c = new TCanvas();
c->Print("canvas.pdf"); //salva la canvas come PDF
c->Print("canvas.C"); //salva la canvas come file .C
c->Print("canvas.root"); //salva la canvas come file ROOT
```

11.9 Salvare l'istogramma o il grafico su un file

```
/* per un grafico è uguale */

TH1F *histo = new TH1F("histo", "maxwell", 100, 0, 15); //crea l'istogramma

TFile *file = new TFile("maxwell.root","RECREATE"); //apre (e crea) il file ROOT
histo->Write(); //scrive l'istogramma sul file
file->Close(); //chiude il file
```

11.10 Leggere l'istogramma o il grafico da un file

```
/* per un grafico è uguale */

TFile *file = new TFile("maxwell.root","READ"); //apre in lettura il file ROOT

TH1F *histo = (TH1F *)file -> Get("histo"); //estrae l'istogramma
histo -> DrawCopy(); //stampa l'istogramma in modo che rimanga

file->Close(); //chiude il file
```

11.11 Eseguire un benchmark del fillaggio istogrammi

```
gRandom -> SetSeed(); //setta il seed

TH1F *h1 = new TH1F("h1", "", 1000, -5., 5.); //
TH1F *h2 = new TH1F("h2", "", 1000, -5., 5.); //crea gli istogrammi e la pdf
TF1 *gaus = new TF1("gaus", "TMath::Gaus(x,0,1)", -5., 5.); //

gBenchmark -> Start("Fill"); //inizia il primo benchmark
for (int i = 0; i < 1e5; i++) {
    Float_t x = gRandom->Gaus(0,1);
    h1 -> Fill(x);
}
gBenchmark -> Show("Fill"); //interrompe e stampa il primo benchmark

gBenchmark -> Start("FillRandom"); //inizia il secondo benchmark
h2->FillRandom("gaus", 1e5);
gBenchmark -> Show("FillRandom"); //interrompe e stampa il secondo benchmark
```

11.12 Funzione definita in una macro

```
Double_t MyFunction(Double_t *x, Double_t *par) { //definisco la funzione nella macro
    Float_t xx = x[0];
    Double_t val = TMath::Abs(par[0]*sin(par[1]*xx)/xx);
    return val;
}

TF1 *f1 = new TF1("f1",MyFunction,0,10,2); //creo la funzione

f1->SetParameters(2,1); //setto i parametri
```


12 ESERCIZI SVOLTI

12.1 Categorie

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 4 bin in un range da 0 a 4, finalizzato a visualizzare il numero di conteggi osservati nelle categorie che contraddistinguono la popolazione di cui a punto successivo.
2. Si genera una popolazione di $N_{TOT}=1e5$ elementi appartenenti a 4 categorie distinte, con rispettive probabilità di occorrenza: caso "0": 1/8, caso "1": 1/4, caso "2": 1/2, caso "3": 1/8 e si riempie l'istogramma con le occorrenze osservate nei diversi casi.
3. si stampano a schermo le frazioni di popolazione osservate nei quattro bin dell'istogramma, con relativa incertezza, usando i metodi espliciti degli istogrammi per estrarre il numero di ingressi totali dell'istogramma, il contenuto dei bin e il relativo errore.

```
void macro() {
    gRandom -> SetSeed();

    TH1F *h = new TH1F("h", "", 4, 0., 4.);

    for(int i = 0; i < 1e5; i++) {
        auto k = gRandom -> Rndm();
        if (k<0.125) {
            h->Fill(0);
        } else if (k<0.375) {
            h->Fill(1);
        } else if (k<0.875) {
            h->Fill(2);
        } else {
            h->Fill(3);
        }
    }

    for (int j = 1; j < 5; j++) {
        std::cout << "Caso " << j << ": " << h->GetBinContent(j) / h->GetEntries()
                    << " +/- " << h->GetBinError(j) / h->GetEntries() << std::endl;
    }

    h -> Draw();
}
```

12.2 PDF data

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 1000 bin in un range da 0 a 5.
2. Si riempie l'istogramma con $1e8$ occorrenze di una variabile casuale x distribuita secondo la p.d.f. (definire esplicitamente una unica funzione che descriva la p.d.f. in esame): $f(x)=x^2/8$ per $0 \leq x < 2$ e $f(x)=0.5$ per $2 \leq x \leq 5$ nel range $[0,5]$, utilizzando il metodo FillRandom della classe di istogrammi.

```
void macro() {
    TH1F *h = new TH1F("histo" , "", 1000, 0., 5.);
    TF1 *f = new TF1("f", "((x*x)/8)*(x<2) + (0.5)*(x>=2)", 0., 5.);

    h -> FillRandom("f", 1e8);
    h -> Draw();
}
```

12.3 Efficienza

Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono 2 istogrammi unidimensionali di 500 bin in un range da 0 a 5.
2. Si riempie il primo istogramma con $1e7$ occorrenze di una variabile casuale x generate esplicitamente e singolarmente (i.e. attraverso `gRandom`) e distribuite uniformemente nel range $[0,5]$ (campione totale).
3. Su tali occorrenze, si simula (attraverso un criterio di reiezione di tipo “hit or miss”) un’efficienza di rivelazione dipendente dalla variabile casuale x secondo la forma: $\text{eff}(x) = e^{-x}$ per $0 \leq x \leq 3$ e $\text{eff}(x) = 0.05$ per $x > 3$.
4. Riempire il secondo istogramma con le occorrenze accettate (campione accettato).
5. Si effettua la divisione fra i due istogrammi per ottenere l’efficienza di rivelazione osservata, utilizzando il metodo `Divide` della classe degli istogrammi e inserendo l’opportuna opzione per la valutazione degli errori secondo la statistica binomiale. Si disegna l’istogramma dell’efficienza visualizzando le incertezze sui contenuti dei bin.
6. Si valuta, usando il metodo che restituisce il numero totale di ingressi, l’efficienza integrale (occorrenze accettate/occorrenze generate)

```
void macro() {
    gRandom -> SetSeed(123);

    TH1F *hgen = new TH1F("hgen", "", 500, 0., 5.);
    TH1F *hacc = new TH1F("hacc", "", 500, 0., 5.);
    hgen -> Sumw2();
    hacc -> Sumw2();

    TF1 *eff = new TF1("eff", "(TMath::E()-x)*(x<=3) + (0.05)*(x>3)", 0., 5.);

    Float_t x, y;
    for(int i = 0; i < 1e7; ++i) {
        x = gRandom -> Uniform(0., 5.);
        hgen -> Fill(x);

        y = gRandom -> Rndm();
        if (y < eff->Eval(x)) {
            hacc -> Fill(x);
        }
    }

    TH1F *hdiv = new TH1F(*hgen);
    hdiv -> Divide(hacc, hgen, 1, 1, "B");

    hdiv -> Draw("E");

    std::cout << hacc -> GetEntries() / hgen -> GetEntries() << std::endl;
}
```

12.4 Somma istogrammi

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono 3 istogrammi monodimensionali di 100 bin in un range da 0 a 5.
2. Si riempie il primo istogramma con $1e4$ occorrenze di una variabile casuale x generate esplicitamente e singolarmente (i.e. attraverso `gRandom`) e distribuite secondo una distribuzione esponenziale decrescente con media = 0.5.
3. Si riempie il secondo istogramma con $1e6$ occorrenze di una variabile casuale x generate esplicitamente e singolarmente (i.e. attraverso `gRandom`) e distribuite secondo una distribuzione gaussiana con media = 2.5 e deviazione standard = 0.25.
4. Si riempie il terzo istogramma con $1e5$ occorrenze di una variabile casuale x generate esplicitamente e singolarmente (i.e. attraverso `gRandom`) uniforme in $[0,5]$.
5. Si fa la somma dei tre istogrammi, e si effettua il fit dell'istogramma risultante secondo una forma funzionale consistente con la distribuzione totale attesa. Si stampano a schermo i parametri corrispondenti alla media e alla deviazione standard della gaussiana che risulta dal fit, con relativo errore, e il chi quadro ridotto.

```
void macro() {
    TH1F *hesp = new TH1F("esp", "", 100, 0., 5.);
    TH1F *hgaus = new TH1F("gaus", "", 100, 0., 5.);
    TH1F *huni = new TH1F("uni", "", 100, 0., 5.);
    hesp -> Sumw2();
    hgaus -> Sumw2();
    huni -> Sumw2();

    for (int i = 0; i < 1e4; i++) {
        auto x = gRandom -> Exp(0.5);
        hesp -> Fill(x);
    }
    for (int i = 0; i < 1e6; i++) {
        auto x = gRandom -> Gaus(2.5, 0.25);
        hesp -> Fill(x);
    }
    for (int i = 0; i < 1e5; i++) {
        auto x = gRandom -> Uniform(0., 5.);
        hesp -> Fill(x);
    }

    TH1F *hsum1 = new TH1F("sum1", "", 100, 0., 5.);
    TH1F *hsum2 = new TH1F("sum2", "", 100, 0., 5.);
    hsum1 -> Add(hesp, hgaus, 1, 1);
    hsum2 -> Add(hsum1, huni, 1, 1);

    TF1 *f = new TF1("fit", "[0]*TMath::Exp(x*[1]) + [2]*TMath::Gaus(x, [3], [4]) + [5]");
    f -> SetParameters(1, 0.5, 1, 2.5, 0.25, 1);

    hsum2 -> Fit(f, "Q");

    auto par = f -> GetParameters();
    auto err = f -> GetParErrors();

    for (int k = 0; k < 6; k++) {
        std::cout << "Parameter " << k << ": " << par[k] << " +/- " << err[k] << std::endl;
    }
    std::cout << "X^2 rid: " << f->GetChisquare() / f->GetNDF() << std::endl;
}
```

12.5 File

Si scriva la parte rilevante e autoconsistente di codice di una macro di ROOT in cui:

1. Si apre un file ROOT (out.root) in modalità di scrittura.
2. Si scrive sul file ROOT un istogramma monodimensionale di Classe TH1D, il cui puntatore e nome dell'oggetto sono rispettivamente h (puntatore) e histo (nome). Si chiude il file root.
3. Si apre il file root in modalità di lettura e si recupera dal file, utilizzando il metodo Get, l'istogramma precedentemente scritto sul file.

```
void macro() {
    TFile *file1 = new TFile("out.root", "RECREATE");
    TH1D *h = new TH1D("histo", "", 10, 0., 1.);

    h -> Fill(0.5);
    h -> Write();
    file1 -> Close();

    TFile *file2 = new TFile("out.root", "READ");
    TH1D *h2 = (TH1D *)file2 -> Get("histo");
    h2->DrawCopy();
    file2 -> Close();
}
```

12.6 Benchmark

Si scriva la parte rilevante e autoconsistente di codice di una macro di ROOT rivolto a monitorare con la classe TBenchmark il tempo di CPU rispettivamente impiegato per fare le due seguenti operazioni, utilizzando anche l'opportuno metodo per stampare a schermo i tempi di esecuzione:

Operazione 1: Generare $1e5$ occorrenze generate esplicitamente e singolarmente di una gaussiana con media = 0 e deviazione standard = 1, riempiendo un istogramma h1 che si assume già opportunamente definito.

Operazione 2: Generare $1e5$ occorrenze di una Gaussiana $G(0,1)$ con il metodo FillRandom, riempiendo un istogramma h2 che si assume già opportunamente definito.

```
void macro() {
    gRandom -> SetSeed();

    TH1F *h1 = new TH1F("h1", "", 1000, -5., 5.);
    TH1F *h2 = new TH1F("h2", "", 1000, -5., 5.);
    TF1 *gaus = new TF1("gaus", "TMath::Gaus(x,0,1)", -5., 5.);

    gBenchmark -> Start("Fill");
    for (int i = 0; i < 1e5; i++) {
        Float_t x = gRandom->Gaus(0,1);
        h1 -> Fill(x);
    }
    gBenchmark -> Show("Fill");

    gBenchmark -> Start("FillRandom");
    h2->FillRandom("gaus", 1e5);
    gBenchmark -> Show("FillRandom");
}
```

12.7 TList

Si scriva la parte rilevante e autoconsistente di codice di una macro di ROOT in cui:

1. Si definiscono in un ciclo 4 istogrammi monodimensionali di tipo TH1D, aventi 1000 bin in un range da 0 a 10, e con nomi root "histo", "histo1", "histo2", "histo3", attraverso un array di puntatori di tipo TH1* e successiva allocazione dinamica degli oggetti.
2. Si impostano i titoli dell'asse x e y rispettivamente a "x variable" e "entries"
3. Si riempie ciascuno degli istogrammi con 10^6 occorrenze di una variabile casuale generate esplicitamente e singolarmente (i.e. attraverso gRandom) secondo delle distribuzioni gaussiane con media, rispettivamente, pari a $\mu = 1, 2, 3, 4$ e deviazione standard $\sigma = 0.1$ (uguale per tutti gli istogrammi).
4. Si inseriscono i quattro istogrammi in un oggetto di tipo TList.
5. In una TCanvas divisa in 4 pad, si disegnano, sempre utilizzando un ciclo, i quattro istogrammi nelle 4 pad distintamente, recuperandoli dalla Tlist e visualizzandoli come linea continua e con le incertezze sui contenuti dei bin.

```
void macro() {
    gRandom -> SetSeed();

    for (int i = 0; i < 4; i++) {
        TH1D *h[i] = new TH1D("histo[i]", "", 1000, 0., 10.);
        h[i] -> SetTitle("", "x variable", "entries");
    }

    for (int i = 0; i < 1e6; i++) {
        auto x0 = gRandom -> Gaus(1., 0.1);
        auto x1 = gRandom -> Gaus(2., 0.1);
        auto x2 = gRandom -> Gaus(3., 0.1);
        auto x3 = gRandom -> Gaus(4., 0.1);
        h0 -> Fill(x0);
        h1 -> Fill(x1);
        h2 -> Fill(x2);
        h3 -> Fill(x3);
    }

    TList *list = new TList()
    for (int i = 0; i < 4; i++) {
        list -> Add(h[i]);
    }

    TCanvas *c = new TCanvas()
    c -> Divide(2, 2);
    for (int i = 0; i < 4; i++) {
        c -> cd(i+1);
        list -> At(i) -> Draw("HISTO", "E");
    }
}
```

ACCANTONAGGIO

Se questa guida ti è stata utile e vuoi offrirmi un vasetto di burro d'arachidi:

<https://www.paypal.me/ricfabbri>

BTC: bc1qa82lu4c8fedz6yh8uq2zplphm536s6pc3vz508

ETH: 0xe5045dd615E7987e199759A57d1Ca78F5a3eDE63