Simulations with ROOT

Thanks to Prof. Massimo Masera for the next slides

Monte Carlo method

- A technique of numerical analysis that uses random sampling to simulate the real-world phenomena
- Applications:
  - Particle physics
  - Quantum field theory
  - Astrophysics
  - Molecular modeling
  - Semiconductor devices
  - Light transport calculations
  - Traffic flow simulations
  - Environmental sciences
  - Financial market simulations
  - Optimization problems
  - ...

What is Monte Carlo Method?

- A method to search for solutions to mathematical problem using a statistical sampling with random numbers.
- This method was developed by Stanislaw Ulam while he committed the hydrogen bomb project at Los Alamos Laboratory after Word War II.
- Historical example of the MC method is Buffon's needle
  - Throw a needle randomly on a sheet on which parallel lines with an equal distance are drawn.
  - Counts the number of throwing which makes the needle crossing the parallel lines.
  - You can get π by random throws.
- An other example one is Laplace's method of calculating π (1886)
  - Area of the square = 4
  - Area of the circle = π
  - Probability of random points inside the circle = π / 4
  - Random points : N
  - Random points inside circle : Nc
  - π ~ 4 Nc / N

Monte Carlo methods for radiation transport

- Fermi (1930): random method to calculate the properties of the newly discovered neutron
- Manhattan project (40's): simulations during the initial development of thermonuclear weapons. von Neumann and Ulam coined the term "Monte Carlo"
- Metropolis (1948) first actual Monte Carlo calculations using a computer (ENIAC)
- Berger (1963): first complete coupled electron-photon transport code that became known as ETRAN
- Exponential growth since the 1980's with the availability of digital computers

Monte Carlo technique

- What is a Random Number?
  - Is e.g. 3 a random number?
  - A sequence of random numbers is a set of numbers that have nothing to do with the other numbers in the sequence
- General procedure for “random” generation:
  - A sequence of m random numbers with uniform distribution in the [0,1] interval is extracted.
  - This sequence is used to produce a second sequence distributed according to a generic f(x), which is the pool of simulated data

Pseudo-Random numbers

- The sequence is reproducible, because the algorithm for the generation is deterministic.
- General characteristic of a random generator:
  - Statistical independence
  - “Long” repetition period
  - The sequence looks random, when indeed it is not

Random generators in ROOT

Are implemented in the **TRandom** class: fast generator with a short (10^9) period, based of the linear congruential method.

- TRandom1: inherits from TRandom and implements RANLUX
- TRandom2: inherits from TRandom has a period of 1026, but only 332 bits word
- TRandom3:inherits from TRandom and implements the Mersenne-Twister generator which has a period of 2^{19937}-1 ($\approx 10^{6002}$).

Here are the CPU times obtained using the four random classes on an lxplus machine with an Intel 64 bit architecture and compiled using gcc 3.4:

	TRandom (ns/call)	**TRandom1** (ns/call)	**TRandom2** (ns/call)	**TRandom3** (ns/call)
`Rndm()`	-	-	6	9
`Gaus()`	31	161	35	42
`Rannor()`	116	216	126	130
`Poisson(m=10)`	147	1161	162	239
`Poisson(m=10)UNURAN`	80	294	89	99

BAD

SLOW

FAST

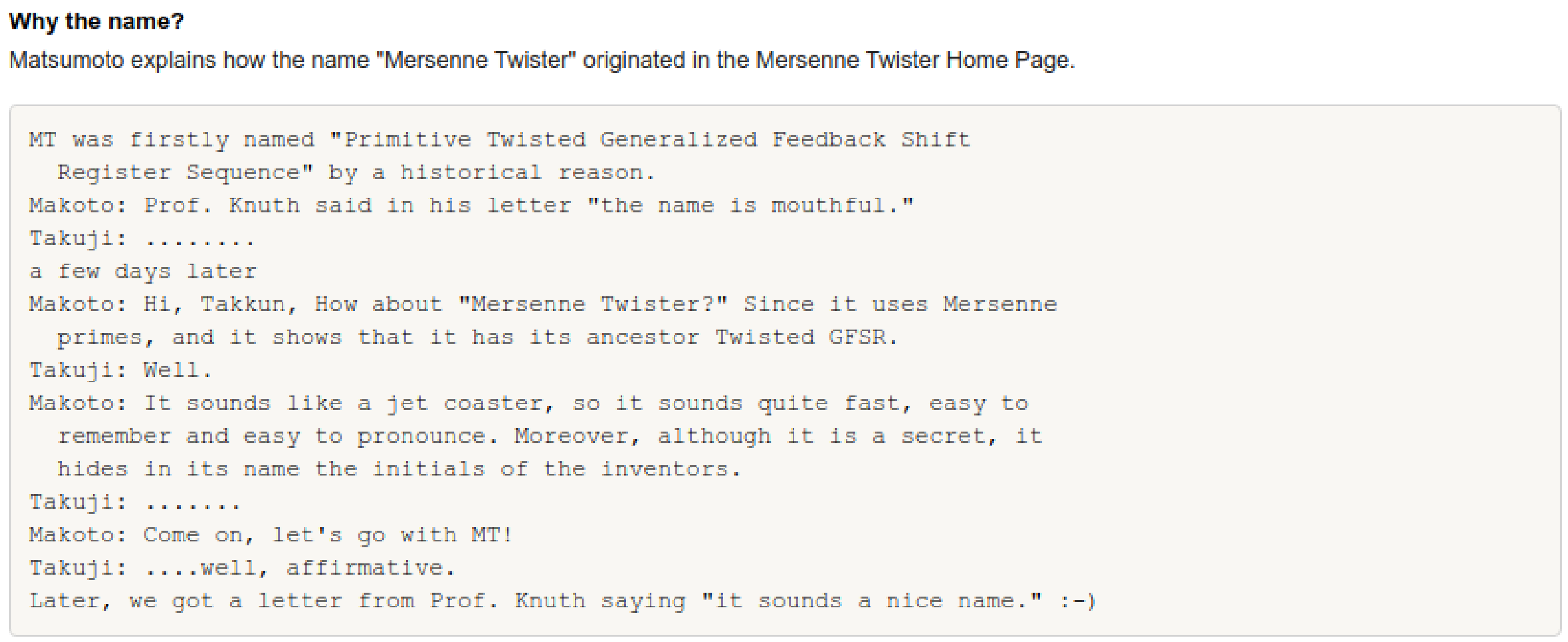
GOOD (default)

Marsenne – Twister generator

- Mersenne Twister is, by far, today's most popular pseudorandom number generator. It is used by every widely distributed mathematical software package. Mersenne Twister was developed by professors Makoto Matsumoto and Takuji Nishimura of Hiroshima University almost twenty years ago.
- **Mersenne primes**
  - Mersenne primes are primes of the form (2^p) - 1 where p itself is prime. They are named after a French friar who studied them in the early 17th century. We learn from Wikipedia that the largest known prime number is the Mersenne prime with p equal to 57,885,161. The Mersenne Twister has p equal to 19937. This is tiny as far as Mersenne primes go, but huge as far as random number generators are concerned.
- **Algorithm**
  - The integer portion of the Mersenne twister algorithm does not involve any arithmetic in the sense of addition, subtraction, multiplication or division. All the operations are shifts, and's, or's, and xor's. All the elements of the state, except the last, are unsigned 32 bit random integers that form a cache which is carefully generated upon startup.
  - This generation is triggered by a seed, a single integer that initiates the whole process. The last element of the state is a pointer into the cache. Each request for a random integer causes an element to be withdrawn from the cache and the pointer incremented. The element is "tempered" with additional logical operations to improve the randomness. When the pointer reaches the end of the cache, the cache is refilled with another 623 elements.
  - The algorithm is analyzed by investigating the group theoretic properties of the permutation and tempering operations. The parameters have been chosen so that the period is the Mersenne prime (2^19937)-1. This period is much longer than any other random number generator proposed before or since and is one of the reasons for MT's popularity.

Marsenne – Twister generator

Why Marsenne – Twister?



Why the name?

Matsumoto explains how the name "Mersenne Twister" originated in the Mersenne Twister Home Page.

```
MT was firstly named "Primitive Twisted Generalized Feedback Shift
  Register Sequence" by a historical reason.
Makoto: Prof. Knuth said in his letter "the name is mouthful."
Takuji: ........
a few days later
Makoto: Hi, Takkun, How about "Mersenne Twister?" Since it uses Mersenne
  primes, and it shows that it has its ancestor Twisted GFSR.
Takuji: Well.
Makoto: It sounds like a jet coaster, so it sounds quite fast, easy to
  remember and easy to pronounce. Moreover, although it is a secret, it
  hides in its name the initials of the inventors.
Takuji: .......
Makoto: Come on, let's go with MT!
Takuji: ....well, affirmative.
Later, we got a letter from Prof. Knuth saying "it sounds a nice name." :-)
```

https://blogs.mathworks.com/cleve/2015/04/17/random-number-generator-mersenne-twister/

Simulation of a radioactive decay of a single nucleus

- Radioactive decay is an intrinsic random process: the probability of decay is constant (independent of the age of nuclei)
- The probability that a nucleus decays in the time Δt is p :

$$p = \alpha \Delta t \;\; (\mathrm{per}\, \alpha \Delta t \ll 1)$$

- Let's consider a system initially having N_0 unstable nuclei: <u>how does the number of nuclei vary with time?</u>

Simulation of a radioactive decay of a single nucleus

- The algorithm which has to be implemented is the following

```
LOOP from t=0 to T, step $\Delta t$
    Reset the number of decayed nuclei ($N_{dec}$) in the current time bin

    LOOP over each remaining parent nucleus (from 0 to $N_{tot}$)
        Decide if the nucleus decays:
        if(gRandom->Rndm()<$\alpha\Delta t$)
            increment the number of decayed nuclei in this time bin
        endif
    END loop over nuclei

    $N_{tot}=N_{tot}-N_{dec}$
    Plot $N_{tot}$ vs t
END loop over time
END
```

Esercitazione 11 – Esercizio 1 (Decay.C)

- Write a macro to implement the algorithm shown in the previous slide. Show the number of remaining nuclei as a function of time for the two following cases:
  - N_0=1000, α=0.01 s^{-1}, Δt=1 s
  - N_0=50000, α=0.03 s^{-1}, Δt=1 s
- Show the results in linear and logarithmic scale, for t between 0 and 300 seconds
- Compare the results with the expected result

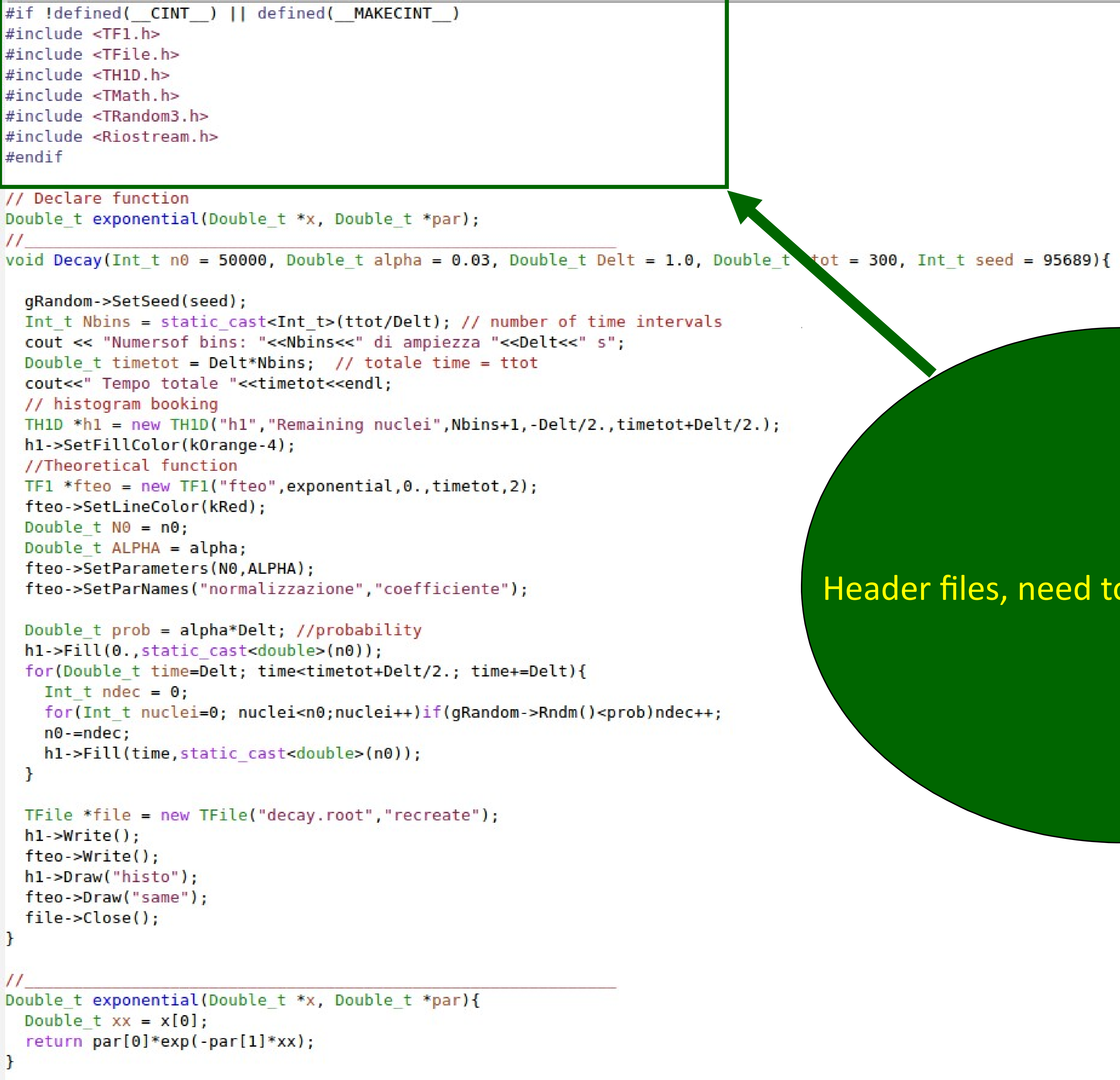
$$dN = -N\alpha dt$$

$$\Downarrow$$

$$N = N_0 e^{-\alpha t}$$

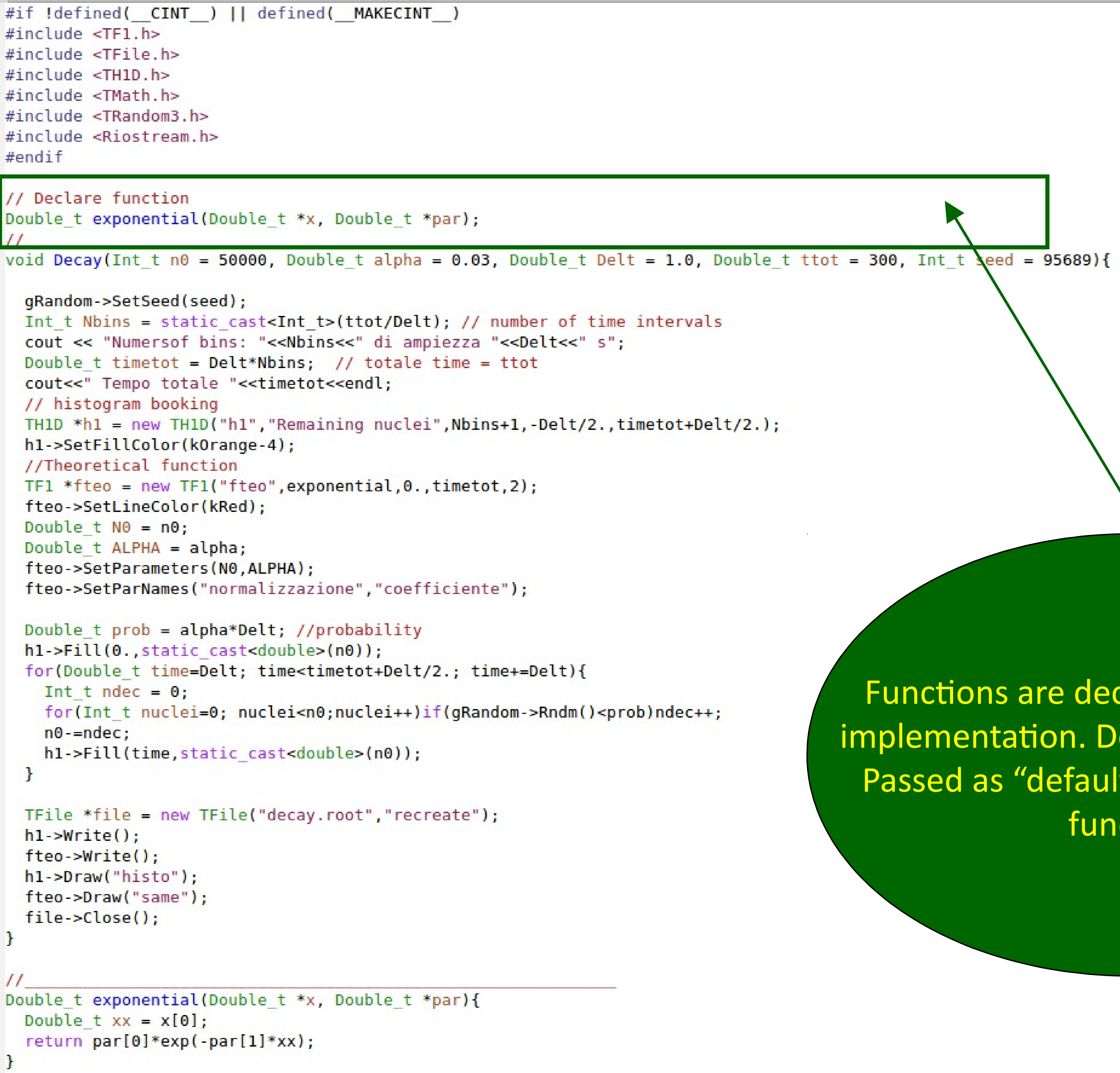
Possible Solution

- In the next slide a possible solution implemented using a macro in ROOT is shown
- The ROOT classes are used to generate random numbers, to store information in the histograms and for input/output operations
- The macro (decay.C) is composed by two functions
- It can be interpret or executed by CLANG (CINT)



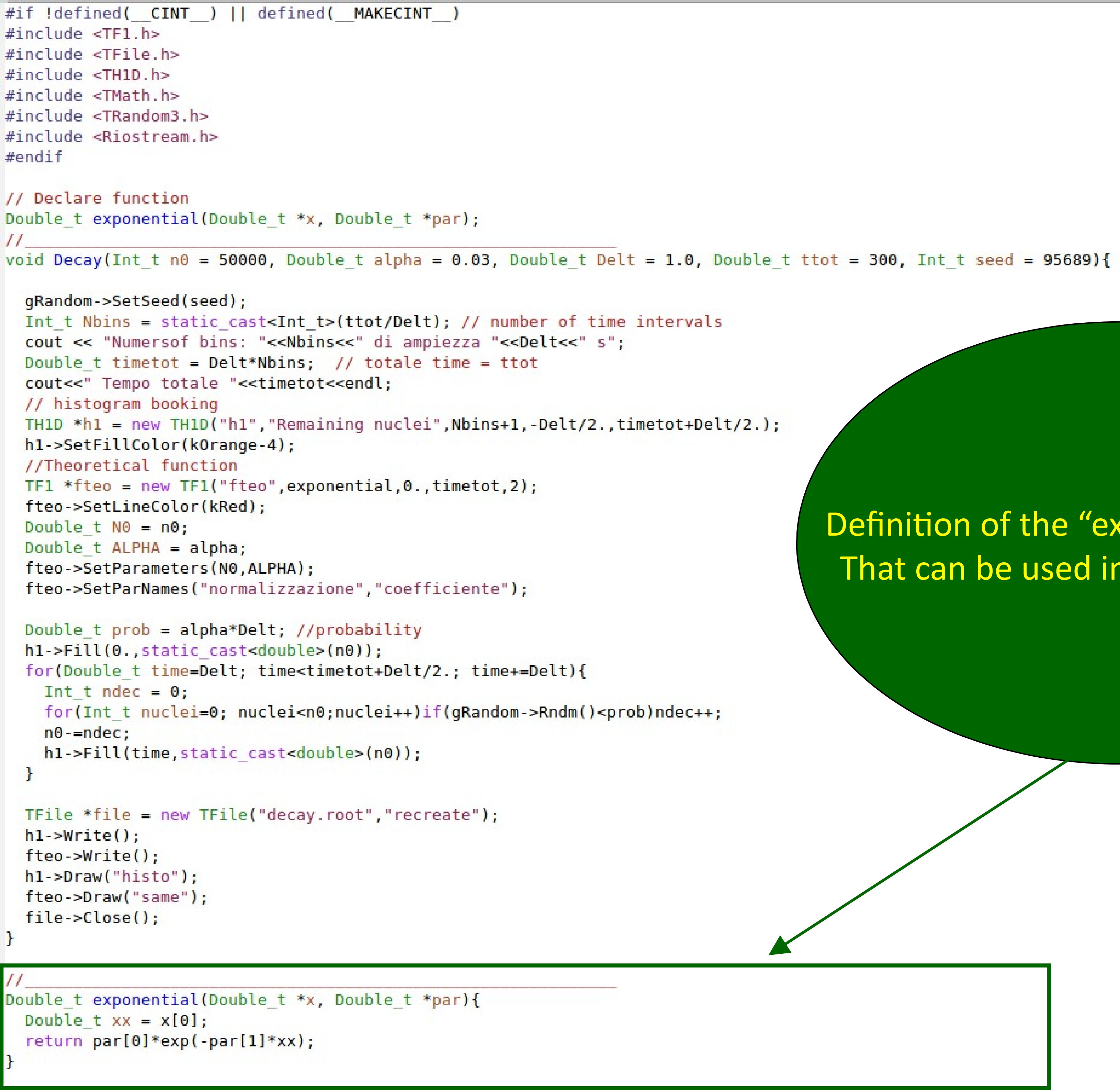
```
#if !defined(__CINT__) || defined(__MAKECINT__)
#include <TF1.h>
#include <TFile.h>
#include <TH1D.h>
#include <TMath.h>
#include <TRandom3.h>
#include <Riostream.h>
#endif

// Declare function
Double_t exponential(Double_t *x, Double_t *par);
//________________________________________________________________
void Decay(Int_t n0 = 50000, Double_t alpha = 0.03, Double_t Delt = 1.0, Double_t ttot = 300, Int_t seed = 95689){

  gRandom->SetSeed(seed);
  Int_t Nbins = static_cast<Int_t>(ttot/Delt); // number of time intervals
  cout << "Numersof bins: "<<Nbins<<" di ampiezza "<<Delt<<" s";
  Double_t timetot = Delt*Nbins;  // totale time = ttot
  cout<<" Tempo totale "<<timetot<<endl;
  // histogram booking
  TH1D *h1 = new TH1D("h1","Remaining nuclei",Nbins+1,-Delt/2.,timetot+Delt/2.);
  h1->SetFillColor(kOrange-4);
  //Theoretical function
  TF1 *fteo = new TF1("fteo",exponential,0.,timetot,2);
  fteo->SetLineColor(kRed);
  Double_t N0 = n0;
  Double_t ALPHA = alpha;
  fteo->SetParameters(N0,ALPHA);
  fteo->SetParNames("normalizzazione","coefficiente");

  Double_t prob = alpha*Delt; //probability
  h1->Fill(0.,static_cast<double>(n0));
  for(Double_t time=Delt; time<timetot+Delt/2.; time+=Delt){
    Int_t ndec = 0;
    for(Int_t nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
    h1->Fill(time,static_cast<double>(n0));
  }

  TFile *file = new TFile("decay.root","recreate");
  h1->Write();
  fteo->Write();
  h1->Draw("histo");
  fteo->Draw("same");
  file->Close();
}

//________________________________________________________________
Double_t exponential(Double_t *x, Double_t *par){
  Double_t xx = x[0];
  return par[0]*exp(-par[1]*xx);
}
```

Header files, need to compile the macro



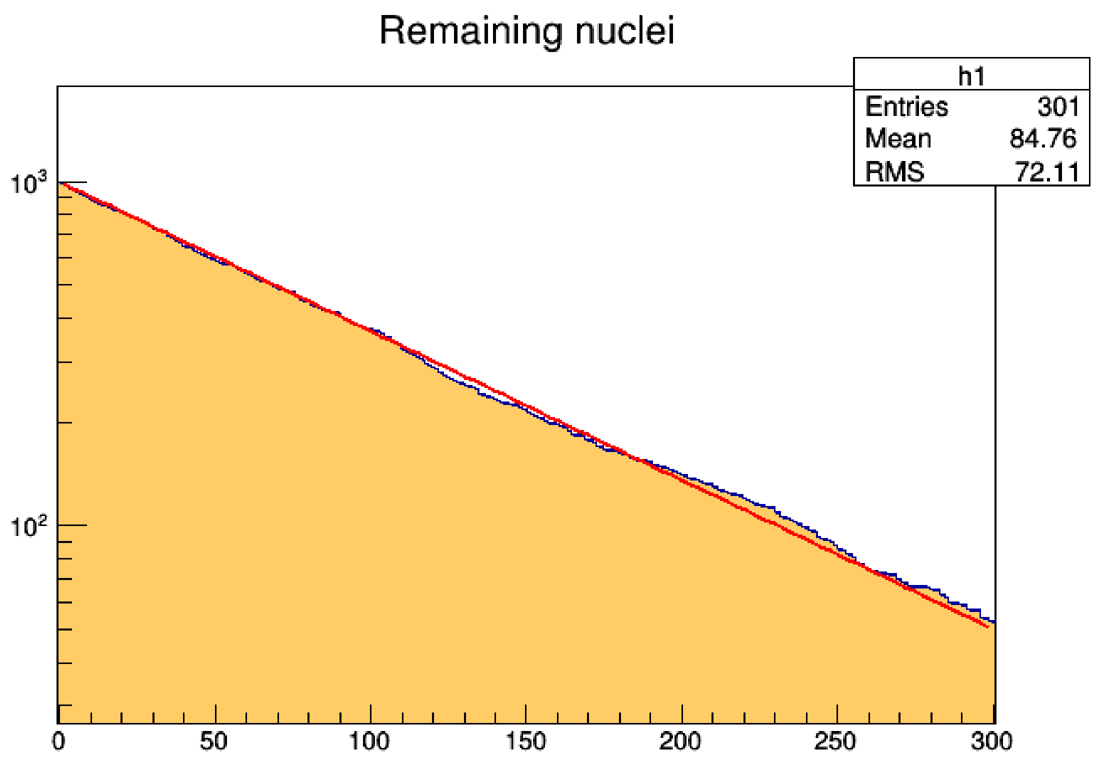
```cpp
#if !defined(__CINT__) || defined(__MAKECINT__)
#include <TF1.h>
#include <TFile.h>
#include <TH1D.h>
#include <TMath.h>
#include <TRandom3.h>
#include <Riostream.h>
#endif

// Declare function
Double_t exponential(Double_t *x, Double_t *par);
//
void Decay(Int_t n0 = 50000, Double_t alpha = 0.03, Double_t Delt = 1.0, Double_t ttot = 300, Int_t seed = 95689){

  gRandom->SetSeed(seed);
  Int_t Nbins = static_cast<Int_t>(ttot/Delt); // number of time intervals
  cout << "Numersof bins: "<<Nbins<<" di ampiezza "<<Delt<<" s";
  Double_t timetot = Delt*Nbins;  // totale time = ttot
  cout<<" Tempo totale "<<timetot<<endl;
  // histogram booking
  TH1D *h1 = new TH1D("h1","Remaining nuclei",Nbins+1,-Delt/2.,timetot+Delt/2.);
  h1->SetFillColor(kOrange-4);
  //Theoretical function
  TF1 *fteo = new TF1("fteo",exponential,0.,timetot,2);
  fteo->SetLineColor(kRed);
  Double_t N0 = n0;
  Double_t ALPHA = alpha;
  fteo->SetParameters(N0,ALPHA);
  fteo->SetParNames("normalizzazione","coefficiente");

  Double_t prob = alpha*Delt; //probability
  h1->Fill(0.,static_cast<double>(n0));
  for(Double_t time=Delt; time<timetot+Delt/2.; time+=Delt){
    Int_t ndec = 0;
    for(Int_t nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
    h1->Fill(time,static_cast<double>(n0));
  }

  TFile *file = new TFile("decay.root","recreate");
  h1->Write();
  fteo->Write();
  h1->Draw("histo");
  fteo->Draw("same");
  file->Close();
}

//____________________________________________________________
Double_t exponential(Double_t *x, Double_t *par){
  Double_t xx = x[0];
  return par[0]*exp(-par[1]*xx);
}
```

Functions are declared before their implementation. Default values can be Passed as “default” argument of the function



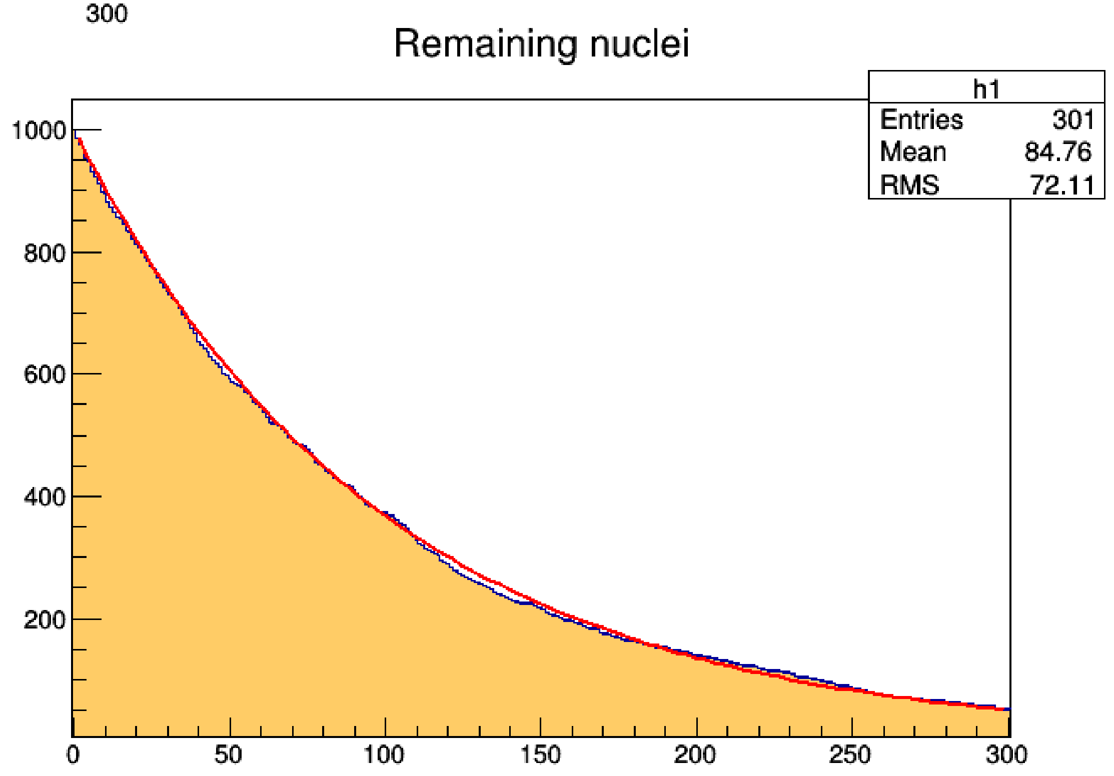
```
#if !defined(__CINT__) || defined(__MAKECINT__)
#include <TF1.h>
#include <TFile.h>
#include <TH1D.h>
#include <TMath.h>
#include <TRandom3.h>
#include <Riostream.h>
#endif

// Declare function
Double_t exponential(Double_t *x, Double_t *par);
//____________________________________________________________
void Decay(Int_t n0 = 50000, Double_t alpha = 0.03, Double_t Delt = 1.0, Double_t ttot = 300, Int_t seed = 95689){

  gRandom->SetSeed(seed);
  Int_t Nbins = static_cast<Int_t>(ttot/Delt); // number of time intervals
  cout << "Numersof bins: "<<Nbins<<" di ampiezza "<<Delt<<" s";
  Double_t timetot = Delt*Nbins;  // totale time = ttot
  cout<<" Tempo totale "<<timetot<<endl;
  // histogram booking
  TH1D *h1 = new TH1D("h1","Remaining nuclei",Nbins+1,-Delt/2.,timetot+Delt/2.);
  h1->SetFillColor(kOrange-4);
  //Theoretical function
  TF1 *fteo = new TF1("fteo",exponential,0.,timetot,2);
  fteo->SetLineColor(kRed);
  Double_t N0 = n0;
  Double_t ALPHA = alpha;
  fteo->SetParameters(N0,ALPHA);
  fteo->SetParNames("normalizzazione","coefficiente");

  Double_t prob = alpha*Delt; //probability
  h1->Fill(0.,static_cast<double>(n0));
  for(Double_t time=Delt; time<timetot+Delt/2.; time+=Delt){
    Int_t ndec = 0;
    for(Int_t nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
    h1->Fill(time,static_cast<double>(n0));
  }

  TFile *file = new TFile("decay.root","recreate");
  h1->Write();
  fteo->Write();
  h1->Draw("histo");
  fteo->Draw("same");
  file->Close();
}

//____________________________________________________________
Double_t exponential(Double_t *x, Double_t *par){
  Double_t xx = x[0];
  return par[0]*exp(-par[1]*xx);
}
```

Definition of the “exponential function”
That can be used in the TF1 definition



Continuous line: expected exponential.

Histogram: simulation result.



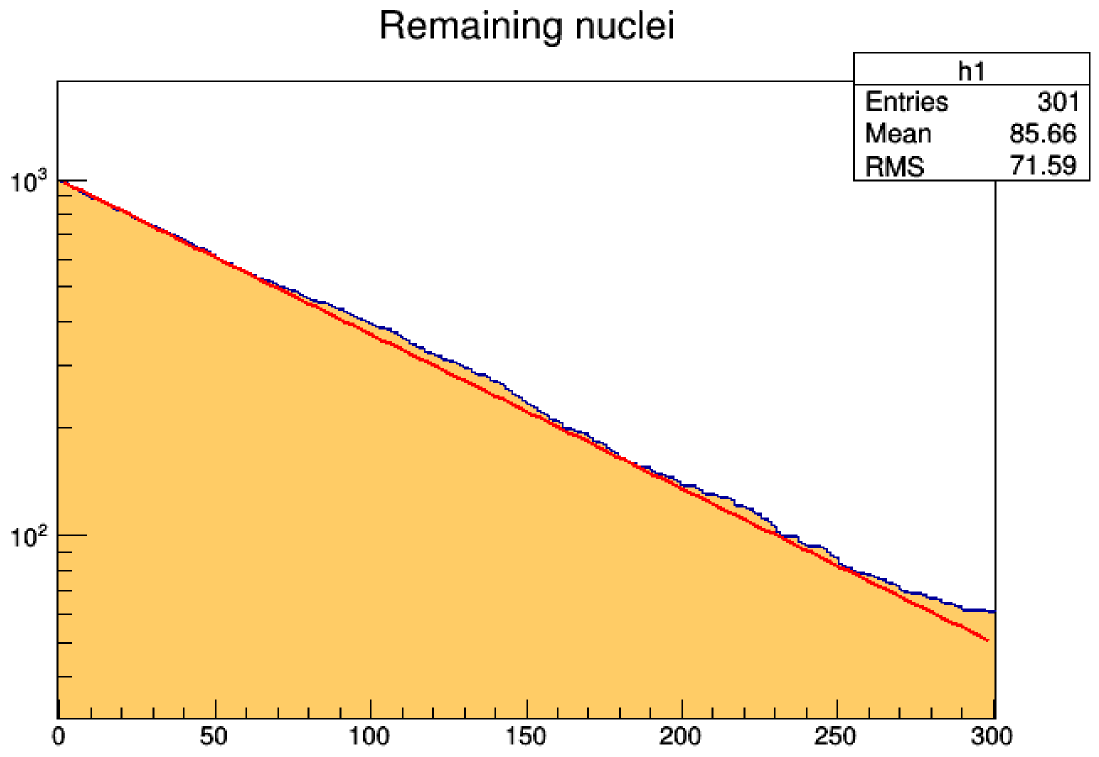
Result for:

N_0=100,

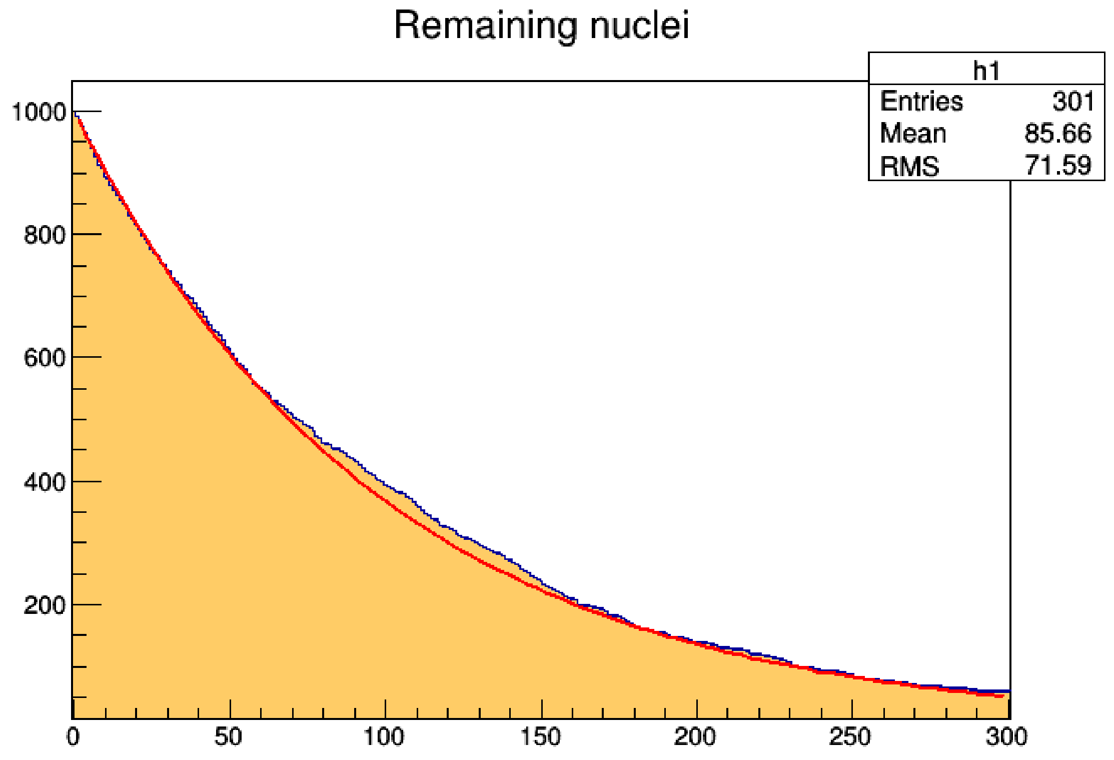
$\alpha=1\times10^{-2}$ s^{-1}

Δt=1 s

Statistical fluctuation are very important

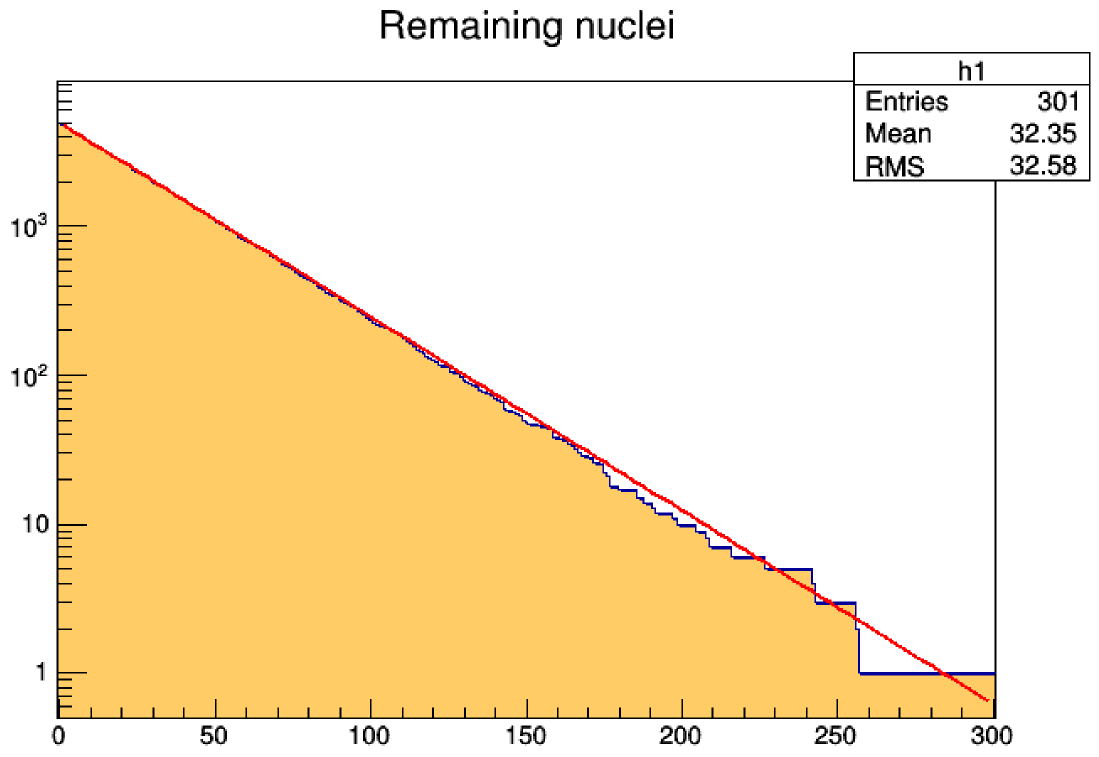


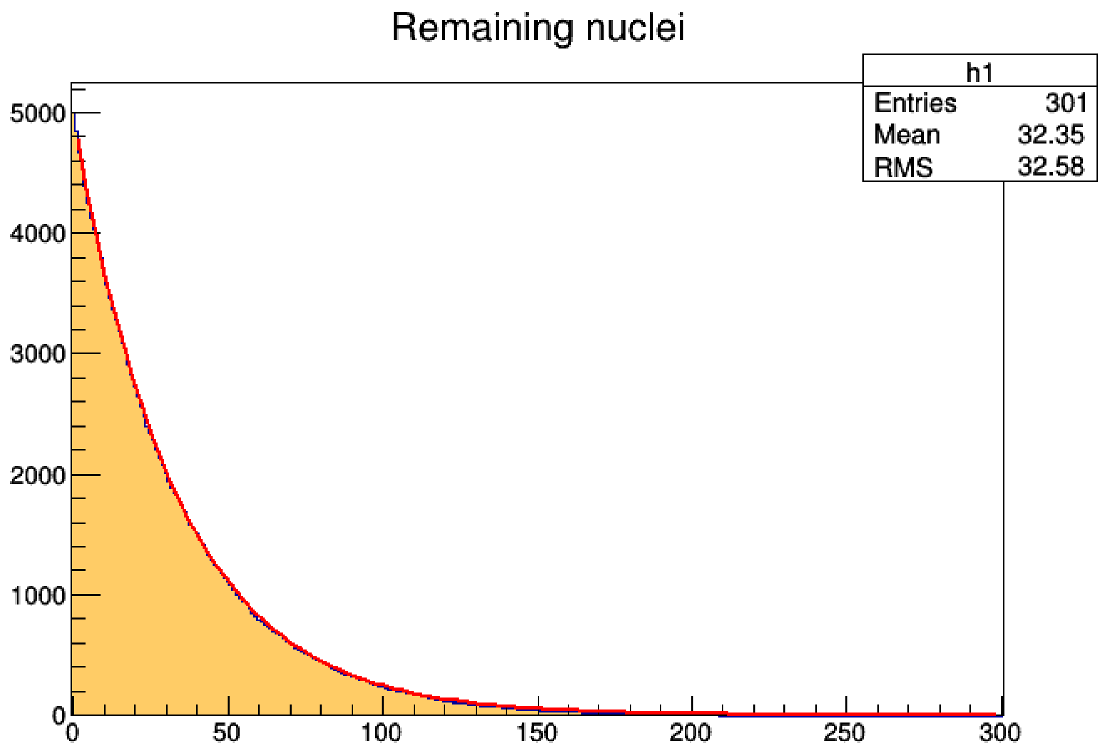
Same parameters as before, but different seed: the results are different!



Risultato per:
$N_0=100$,
$\alpha=1\times10^{-2}\ s^{-1}$
$\Delta t=1$ s

The importance of fluctuations depends on the number N of residual nuclei

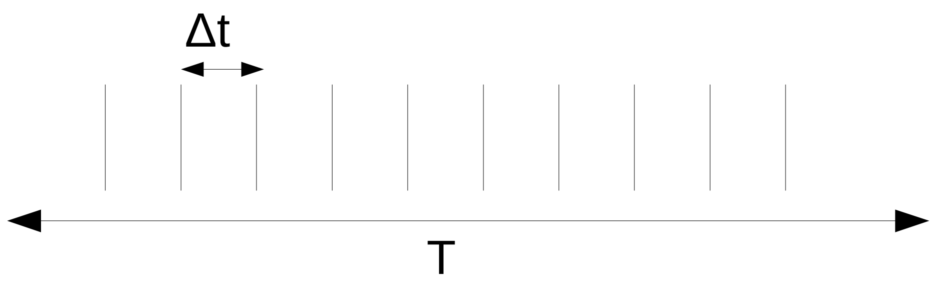




Result for:
N_0=5000,
$\alpha=3\times10^{-2}$ s^{-1}
Δt=1 s

How many decays in a fixed time interval T?

- Let's assume the number of decays in time T is much less than the number of parent nuclei. (i.e. let's assume a constant probability to observe a decay)
- The time T can be break into *m* shorter intervals of duration Δt (T/m):



- The probability to observe 1 decay in time Δt is :

$$p = \beta \Delta t$$

How many decays in a fixed time interval T?

$$p = \beta \Delta t$$

- β = αN (Note that α is the decay probability)
- Δt must be small enough so that βΔt << 1
- The probability of observing n decays in time T is given by the binomial distribution:

$$\boxed{P(\text{n decays in m intervals}) = \binom{m}{n} p^n (1-p)^{(m-n)}}$$

$$\boxed{P = \frac{m!}{n!(m-n)!} \left(\frac{\beta T}{m}\right)^n \left(1 - \frac{\beta T}{m}\right)^{(m-n)}}$$

How many decays in a fixed time interval T?

- In the limit of large m (m→ ∞ and Δt → 0)

$$\left.\begin{aligned} &\left(1-\frac{\beta T}{m}\right)^{m} \rightarrow e^{-\beta T} \\ &\left(1-\frac{\beta T}{m}\right)^{-n} \rightarrow 1 \\ &\frac{m!}{(m-n)!} \rightarrow m^{n} \end{aligned}\right\}$$

$$\boxed{P(n;\beta T)=\frac{(\beta T)^{n}}{n!}e^{-\beta T}=\frac{\mu^{n}e^{-\mu}}{n!}}$$

Known as Poisson distribution

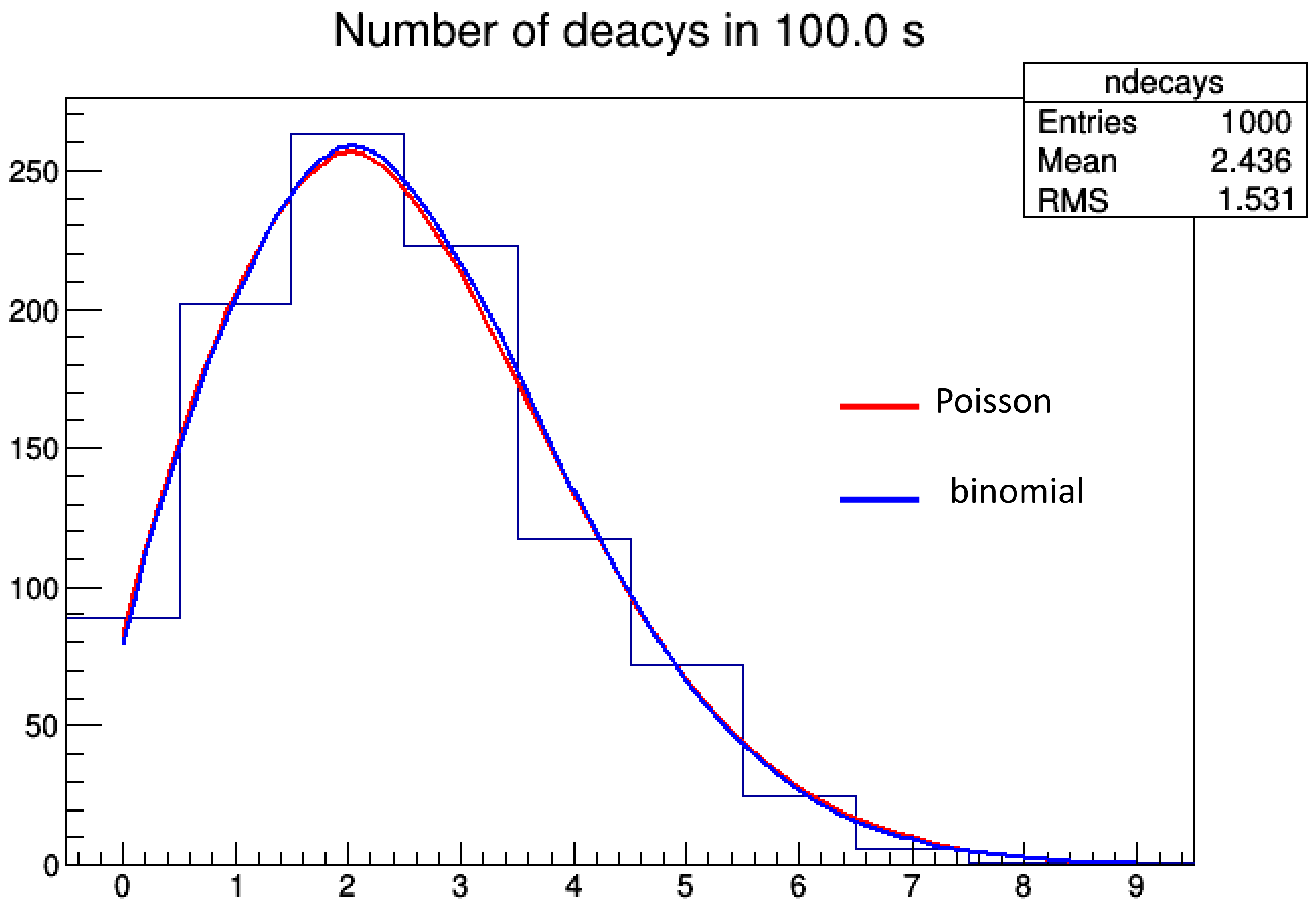
Esercitazione 11 – Esercizio 2 (poisson.C)

- Modify the program written for exercise 1 to simulate an experiment that counts the number of decays observed in a time interval, T .
- Allow the experiment to be repeated and represent in a histogram the distribution of the number of decays for the following two cases:

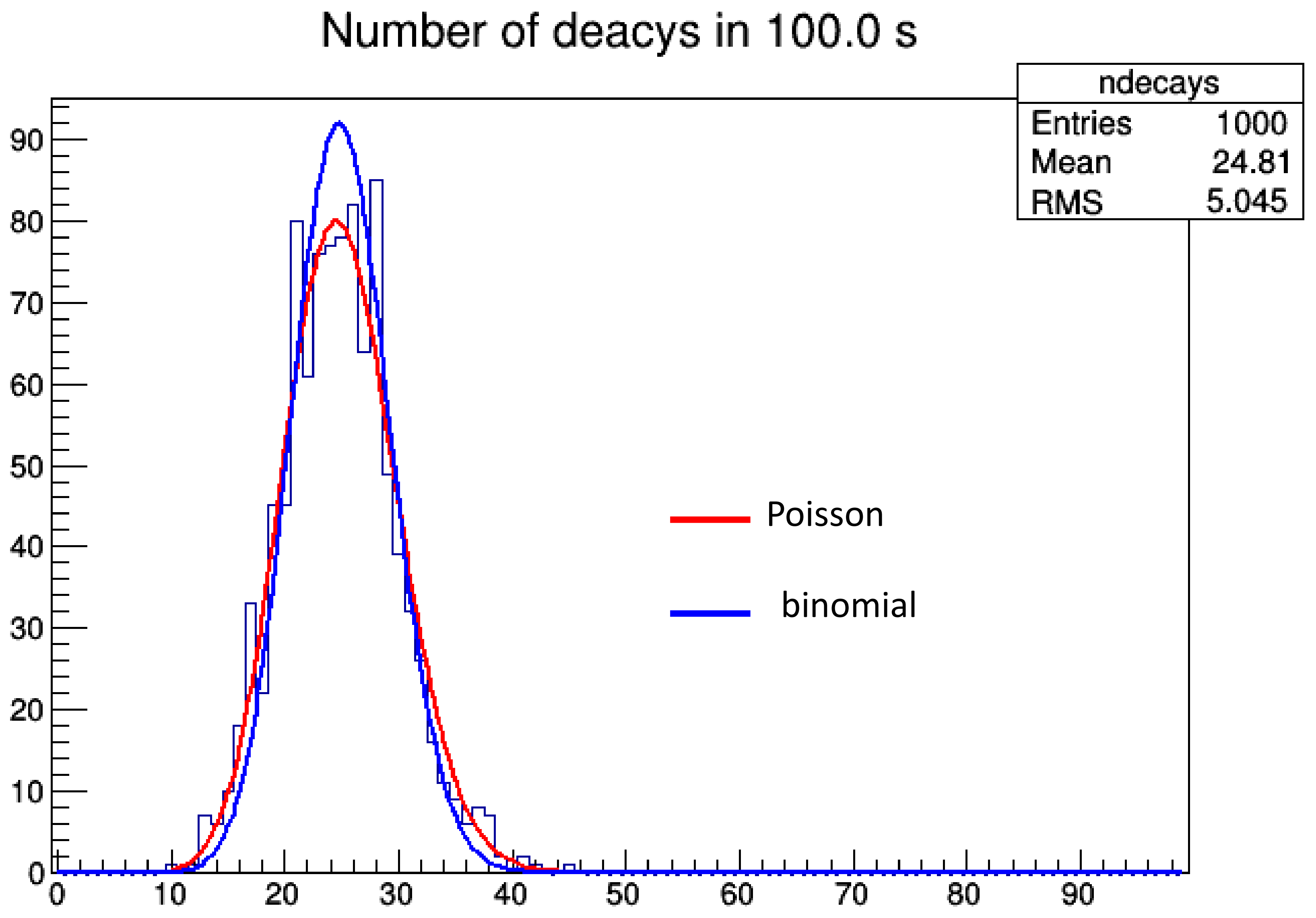
  N_0=1000, α=2.5x10^{-5} s^{-1}, Δt=1 s, T = 100s

  N_0=1000, α=2.0x10^{-4} s^{-1}, Δt=1 s, T = 100s
- Compare the distribution of n for 1000 experiments with Binomial and Poisson distributions.

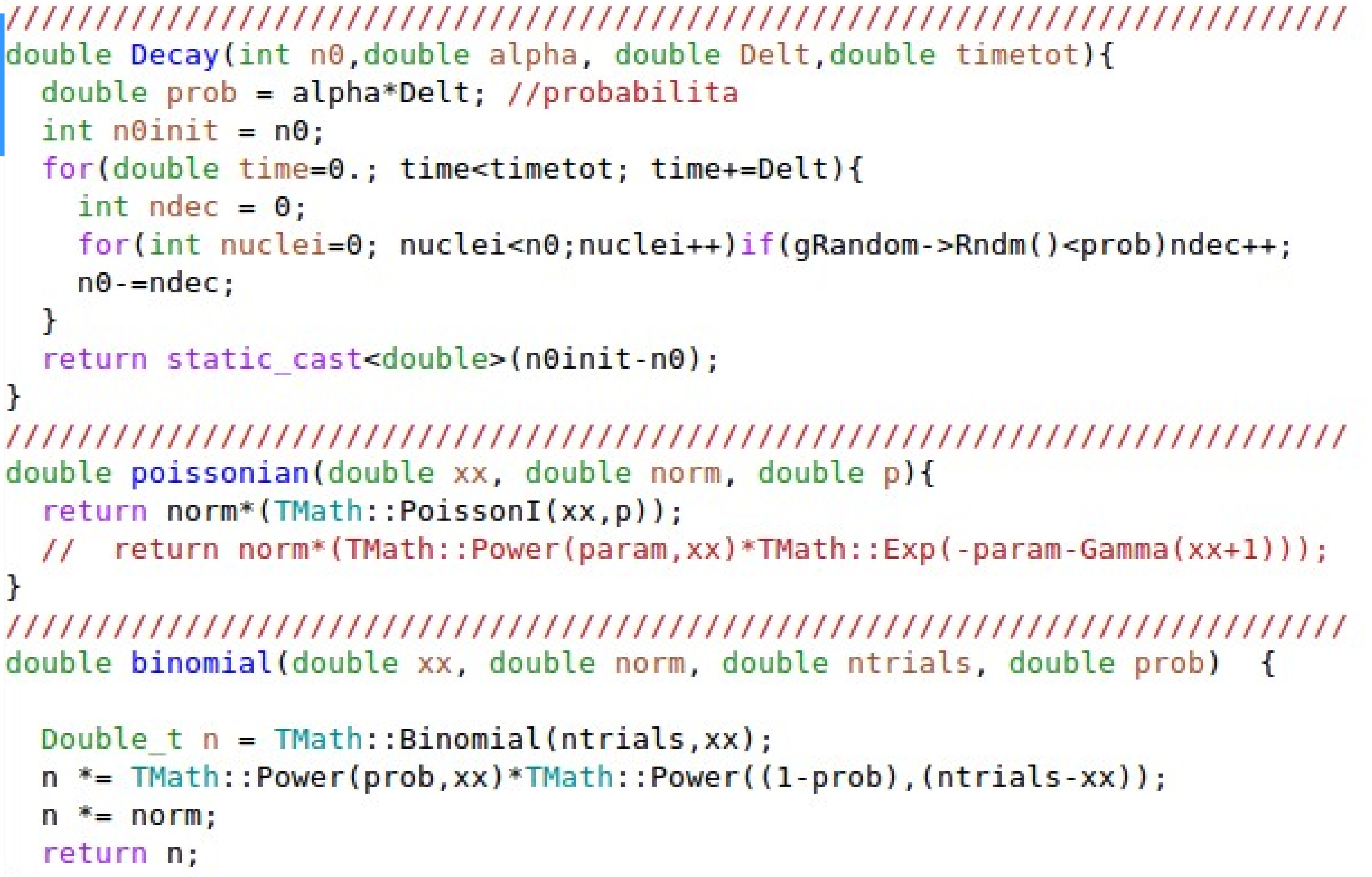
Result for: N_0=1000, α=2.5X10^{-5}, Δt=1 s, T=100 s



Result for: N_0=1000, α=2.5X10^{-4}, Δt=1 s, T=100 s



```
// Function declaration
double Decay(int n0,double alpha, double Delt,double timetot);  // Decay simulation

double poissonian(double xx, double norm, double p); // POISSONIAN density
double binomial(double xx, double norm, double ntrials, double prob);   // BINOMIAL density
void poisson(int n0 = 1000,double alpha = 2.5e-5, double Delt = 1.,double time=100.,unsigned int seed = 1234); //main function

///////////////////////////////////////////////////////////////////////////
void poisson(int n0,double alpha, double Delt,double time,unsigned int seed){

  gRandom->SetSeed(seed);

  TString title = Form("Number of deacys in %4.1f s",time);
  int nbins = static_cast<int>(n0*alpha*time*4);
  cout<<"numer of bins"<<nbins<<endl;
  double high = nbins-0.5;
  TH1F *ndecays = new TH1F("ndecays",title,nbins,-0.5,high);

  //theoretical function (poissonian)
  double mu = alpha*n0*time;
  title =Form("Poissonian distribution with parameter %10.4g ",mu);

  TH1F *fteo = new TH1F("fteo",title,nbins,-0.5,high);

  //theoretical function 2(binomial)
  double ntrials = time/Delt;
  double prob = alpha*n0*Delt;
  title = Form("Binomial with parametris p= %10.4g e m = %8.2g",prob,ntrials);

  TH1F *fteo2 = new TH1F("fteo2",title,nbins,-0.5,high);

  cout<<"Numer of nuclei: "<<n0<<endl;
  cout<<"Parameter of poissonian distribution "<<mu<<endl;
  cout<<"Parameters of binomial distribution p= "<<prob<<" m ="<< ntrials<<endl;


  // Fill histo with theoretical functions
  for (double x=0.;x<high;x+=1.)
    fteo->Fill(x,poissonian(x,knorm,mu));
  for (double x=0.;x<high;x+=1.)
    fteo2->Fill(x,binomial(x,knorm,ntrials,prob));

  // Simulation of the decay
  for(int exper = 0; exper<knorm; exper++){
    if((exper%100)==0)cout<<"Processinf event n. "<<exper<<endl;
    double nodecay=Decay(n0,alpha,Delt,time);
    ndecays->Fill(nodecay);
  }
```



```
///////////////////////////////////////////////////////////////////////////
double Decay(int n0,double alpha, double Delt,double timetot){
  double prob = alpha*Delt; //probabilita
  int n0init = n0;
  for(double time=0.; time<timetot; time+=Delt){
    int ndec = 0;
    for(int nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
  }
  return static_cast<double>(n0init-n0);
}
///////////////////////////////////////////////////////////////////////////
double poissonian(double xx, double norm, double p){
  return norm*(TMath::PoissonI(xx,p));
  //  return norm*(TMath::Power(param,xx)*TMath::Exp(-param-Gamma(xx+1)));
}
///////////////////////////////////////////////////////////////////////////
double binomial(double xx, double norm, double ntrials, double prob)  {

  Double_t n = TMath::Binomial(ntrials,xx);
  n *= TMath::Power(prob,xx)*TMath::Power((1-prob),(ntrials-xx));
  n *= norm;
  return n;
```