# 4-й семестр, lesson2.
# ROOT and 1-dimensional histograms.

- - To start a ROOT session, just type “ root “ or “ root –l “. Command “.help” provides some help. To quit the ROOT session, type “.q “ . If the interface hangs, type .qqqqq
- - ROOT has a powerful C/C++ interpreter giving you access to all available ROOT classes, global variables, and functions via the command line (GUI).
- - By typing C++ statements at the prompt, you can create objects, call functions, execute scripts, etc
- - For example, type[ root] 1+sqrt(9)
- - User can type commands one after another, or write a file with commands enclosed by {...}
- - The "include" statements may be needed at the beginning, like
- #include <iostream> for input/output
- #include "TMath.h”
- #include "TH1"
- An example: from command line :
- int i=2; int j=4;
- cout << " i*j = " << i*j << endl;

## Getting started – ROOT commands

- Write in file  example.C
- double bb_px;
- double bb_py;
- double bb_phi;
- bb_px = 1.;
- bb_py = 1.;
- bb_phi = TMath::ATan2(bb_py, bb_px);
- cout << " TMath::ATan2(bb_py, bb_px) = " << bb_phi << endl; }
- Then start the ROOT and type .X  example.C
- .q

# ROOT User’s guide

- https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html
- •Chapters:
- 1.Introduction
- 2.Getting Started
- 3.Histograms
- 4.Graphs
- 5.Fitting Histograms
- 6.A Little C++
- 7.ClNT the C++ Interpreter
- 8.Object Ownership
- 9.Graphics  and the Graphical User Interface
- 10.Folders and Tasks
- 11.Input/Output
- l12Trees  ( to be continued
- 13. Math Libraries in ROOT ( and more …

## User’s Guide in PDF

- Histograms – see
- https://root.cern.ch/root/html534/guides/users-guide/ROOTUsersGuide.html#histograms

- Operations by mouse
- And from the command line
- Commands can be written to file
-   vim example.C
- And then executed in ROOT by sequence :  .example.C

## Work with 1-dimensional histogram

- Create object
- Indicate details
- Fill histogram
- Draw histogram
- Write it to output file

# 1-dimensional histograms

- Booking  a histogram:  static TH1D * h1;
- int Nbins = 50;
- double Xlow = 0.;
- double Xhigh= 100.;
- h1 = new TH1D("h1", "title", Nbins, Xlow, Xhigh);
- filling: in the loop:
- double value;
- // value = f(x);
- h1 -> Fill( value );
- show histogram  after filling:  h1 -> Draw();
- .q

# Input/output and weighted histograms

- open **output** file and write histogram:
- TFile *_filewOut= TFile::**Open("./**ntuple.root", "recreate");
- **h1 -> Write();**
- _filewOut -> **Close();**
- open **inpu**t file:
- TFile *_file0 = TFile::Open("input_file.root"  ;

- some details:
- 1)
- if **weighted** histogram needed,
- then the error on bin content is calculated
- as sqrt ( sum of weight*weight );
- a supplementary statement needed just after histogram definition,
- h1 -> Sumw2();
- and the weight value should be supplied  for event:
- **h1 -> Fill( value, weight );**

# Canvas and Pads

- 2) The basic whiteboard in which an object is drawn in ROOT is called a **canvas** (defined by the class **TCanvas**). Every object in the canvas is a graphical object in the sense that you can grab it, resize it, and change some characteristics using the mouse.
- The canvas area can be divided in several sub areas, so-called **pads** (the class **TPad**). A pad is a canvas sub area that can contain other pads or graphic objects.
- At any one time, just one pad is the so-called active pad. Any object at the moment of drawing will be taken in the active pad.
- The obvious question is what is the relation between a canvas and a pad? In fact, a canvas is a pad that spans throughout the entire window. This is

  nothing else than the notion of inheritance. The **TPad** class is the parent of

  the **TCanvas** class.
- In ROOT, most objects derive from a base class **TObject**. This class has a virtual method Draw() such as all objects supposed to be "drawn". If several canvases are defined, there is only one active at a time.
- One draws an object in the active canvas by using the statement: **object.Draw**(). This instructs the object "object" to be drawn.

# Objects and directories

- If no canvas is opened, a default one (named "**c1**" is created.
- The following command does a subdivision of canvas to pads:
- **c1 -> Divide(3, 2**). It creates 6 pads ( 3 on X by 2 on Y dimension).
- Canvas can be cleaned from previous objects: **c1 -> Clean().**
- Activation of a pad is performed by command **c1->cd(1),** ..., c1->cd(6).
- After filling, a canvas can be written to file in different formats:
- **c1 -> SaveAs**("file_name", "pdf")
- c1 -> SaveAs("file_name", "png")
- c1 -> SaveAs("file_name", "eps").
- 3) Histograms and other objects can be arranged in directories,
- like usual files. The “ **.ls**” command shows content of current directory, "cd" command provides change the current directory.

# Various comments

- 4) Objects can be written to **file** or kept in **memory**, and in some cases this difference  does  matter.
- The  “.ls “ command **.ls** marks as "**KEY**" the object in file  and the objects in memory as "**OBJ**".
- 5) There are **tips** in histograms, canvases and other classes, which indicate possible actions with object, in the same spirit as LINUX provides tips for file names after indication of  first characters and then pushing the TAB.
- 6) **Warning:** if the commands are written in file and it  has structure like
- #include "TH1"
- void example(){
- ...
- }
- then the **name of  the file** should be the same as the **name of the main** segment, i.e. both should be  called " example ".

# Exercise 1

- **Log-in** to your work directory
- Copy files to you r work directory
- **scp –p /nfs/lfi.mipt.su/data/student/2021/2021_0215/**example_2_edited.C .
- scp –p /data/nikola/ves/run42/ntbeam_cher_r17_1_v15.root . .
- Or create a link to this data file :
- ln –s /nfs/lfi.mipt.su/data/nikola/ves/run42/ntbeam_cher_r17_1_v15.root ntbeam_cher_r17_1_v15.root
- It contains NTuple with data file and an example of a program in file "**example_2_edited.C**"
- It **reads** the input Ntuple, gets the number of reconstructed $\pi^0$ in event and **fills** a corresponding histogram.
- Try to start ROOT and run the ROOT command file:
- **root**
- **.X example_2_edited.C**
- .ls
- The identifier of the histogram is h_001 . Before leaving the ROOT, one can do several exercsises:
- cout << **h_001->GetEntries()** << endl;
- cout << **h_001->GetBinContent(1)** << endl;

# Exercise 1

- cout << **h_001->GetBinContent(2)** << endl;
- c1->**SetLogy();**
- c1->SetLogy(0);
- cout << h_001->GetBinError(2) << endl;
- h_001->GetXaxis()->SetTitle("Mass, GeV");
- h_001->SetLineColor(kRed); _filewOut->Close();
- h_001->Draw();
- .q
- Open Output file and write the h_001 into output file.
- Add Write:

```
  TFile *_filewOut= TFile::Open("output.root", "RECREATE");
   h_001->Write();
    _filewOut->Close();
```

Then : root -l output.root

```
        h_001->Draw();
       c1->SaveAs("ouput.pdf", "pdf");
      .q
```