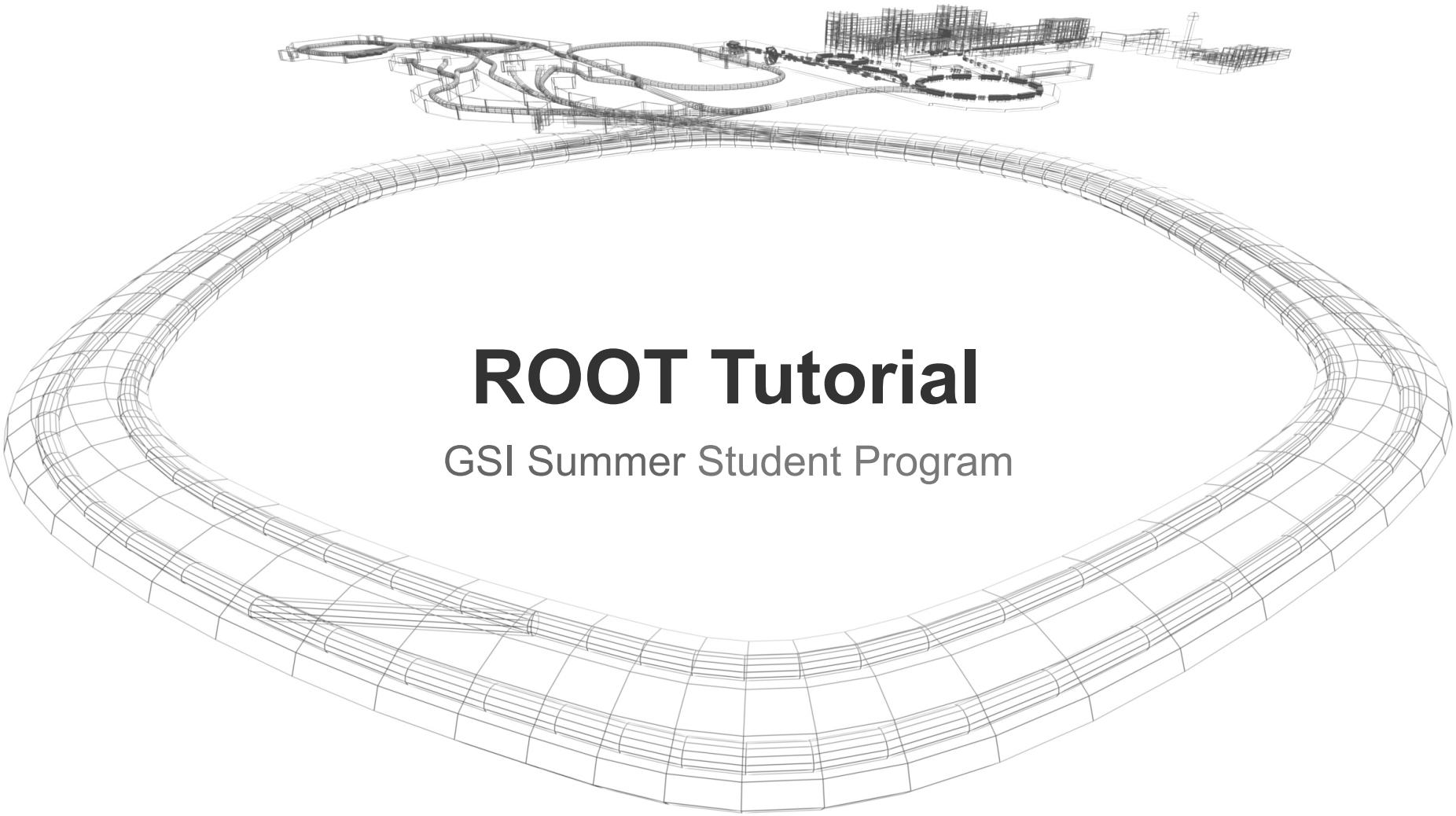
GSI Helmholtzzentrum für Schwerionenforschung GmbH



ROOT Tutorial

GSI Summer Student Program

Questionnaire

- Have you heard about ROOT before?

Questionnaire

- Have you heard about ROOT before?
- Have you used ROOT before?

Questionnaire

- Have you heard about ROOT before?
- Have you used ROOT before?
- Have you used C++ before?

Questionnaire

- Have you heard about ROOT before?
- Have you used ROOT before?
- Have you used C++ before?
- Have you used any other programing language before?

Questionnaire

- Have you heard about ROOT before?
- Have you used ROOT before?
- Have you used C++ before?
- Have you used any other programing language before?
- Have you used Linux before?

Questionnaire

- Have you heard about ROOT before?
- Have you used ROOT before?
- Have you used C++ before?
- Have you used any other programing language before?
- Have you used Linux before?
- Have you worked on the Linux command line before?

Outline

- Introduction
- Start using ROOT
  - working at the ROOT prompt
  - do some simple calculations
  - creating and plotting functions
- Work with data values
  - plot some measurements
  - describe the data with a model
- Histograms in ROOT
  - histogram operations
  - multi-dimensional histograms
  - display options and visualization in ROOT
- I/O, Ntuples andTrees
  - how to create and fill trees and how to read and analyze

This Tutorial

- You can find this presentation and other material in a git repository at https://github.com/fuhlig1/SummerStudent_ROOT_2017
- During the course I will add more material to the repository
  - Sample solutions for the exercises
  - ROOT macros
- Get the presentation and other material to your computer
  - Open a terminal using the terminal desktop icon
  - Execute the following command on the terminal
    - git clone https://github.com/fuhlig1/SummerStudent_ROOT_2017
- Update the repository (needed later)
  - Enter the directory which contains the ROOT tutorial
  - Type in the terminal the following command
    - git pull origin

This Tutorial

- This is an introductory tutorial
  - Maybe boring if you have already used ROOT
- Objectives
  - Get some basic ideas about C++
  - Become familiar with the ROOT toolkit
  - Be able to use the ROOT prompt
  - Plot data
  - Describe the data with a model (Fit the data)
  - Perform basic I/O operations
- Idea
  - Slides introduce the most important features and concepts
  - Hands on exercises to learn how to use ROOT

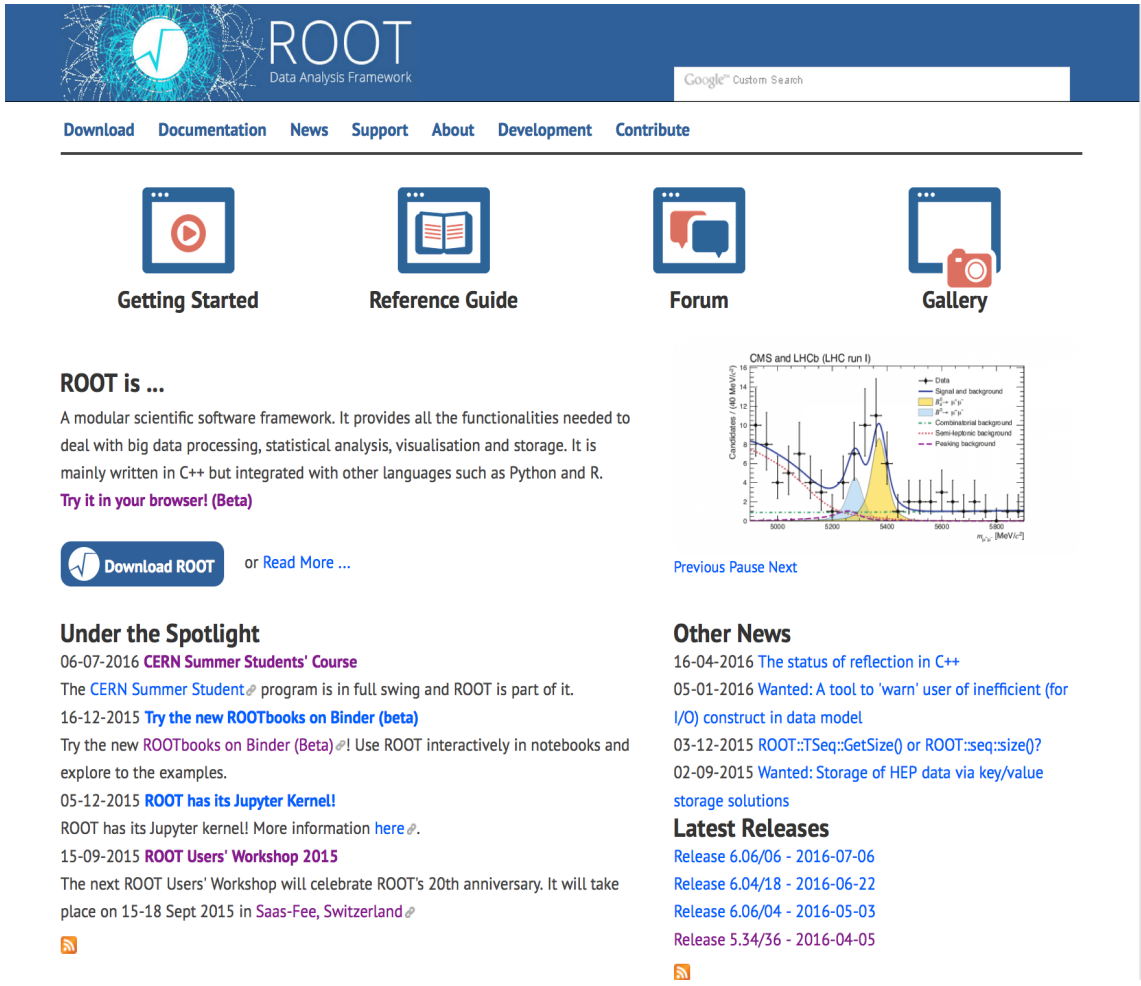
This Tutorial

- There are several options to use ROOT for the hands on part

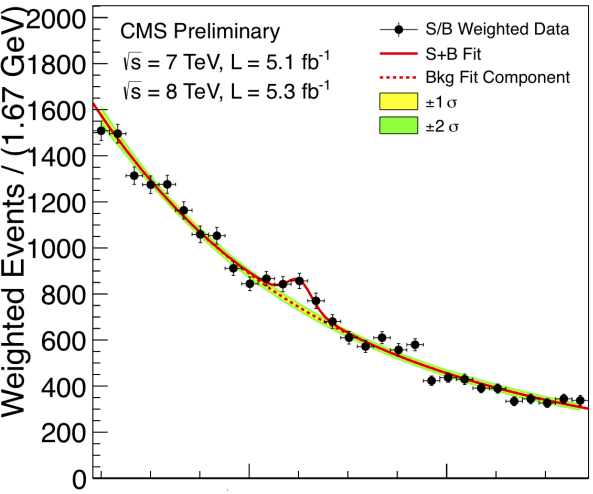
1. Use the GSI installation of ROOT
   - Needs GSI Linux account
   - There are training accounts for the computers in this room
2. Local ROOT installation on your own computer
   - https://root.cern.ch/downloading-root
3. Use a virtual machine with preinstalled ROOT
   - Installation instruction at https://github.com/fuhlig1/SummerStudent_ROOT_2017/blob/master/README.md

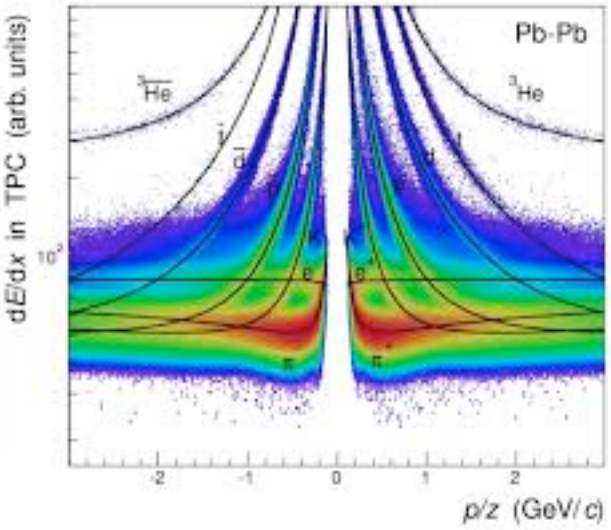
Sources of Information

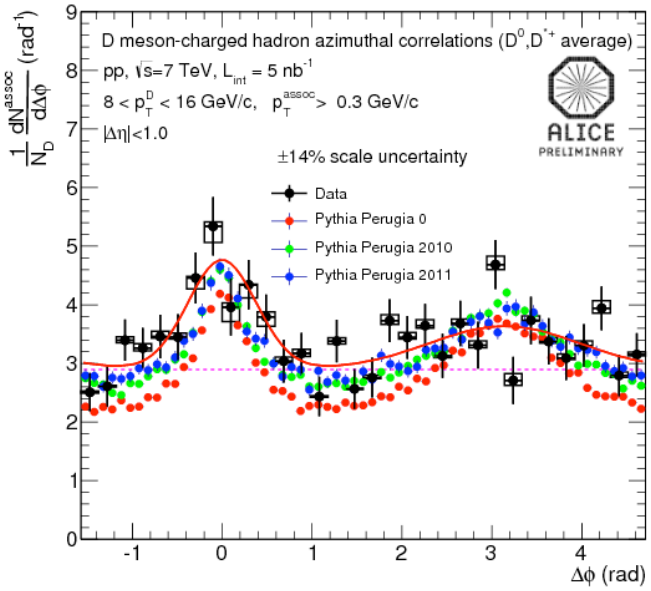
- ROOT webpage (root.cern.ch)
  - Download and installation instructions
  - Class documentation
  - Manuals
  - Tutorials
  - Presentation
  - Forum
  - …
- Check the web site during the first hands on session

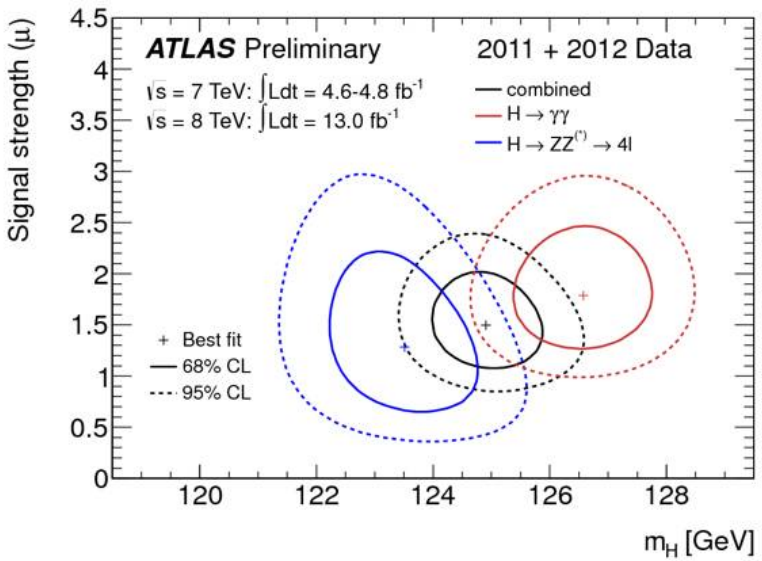


What is ROOT good for?



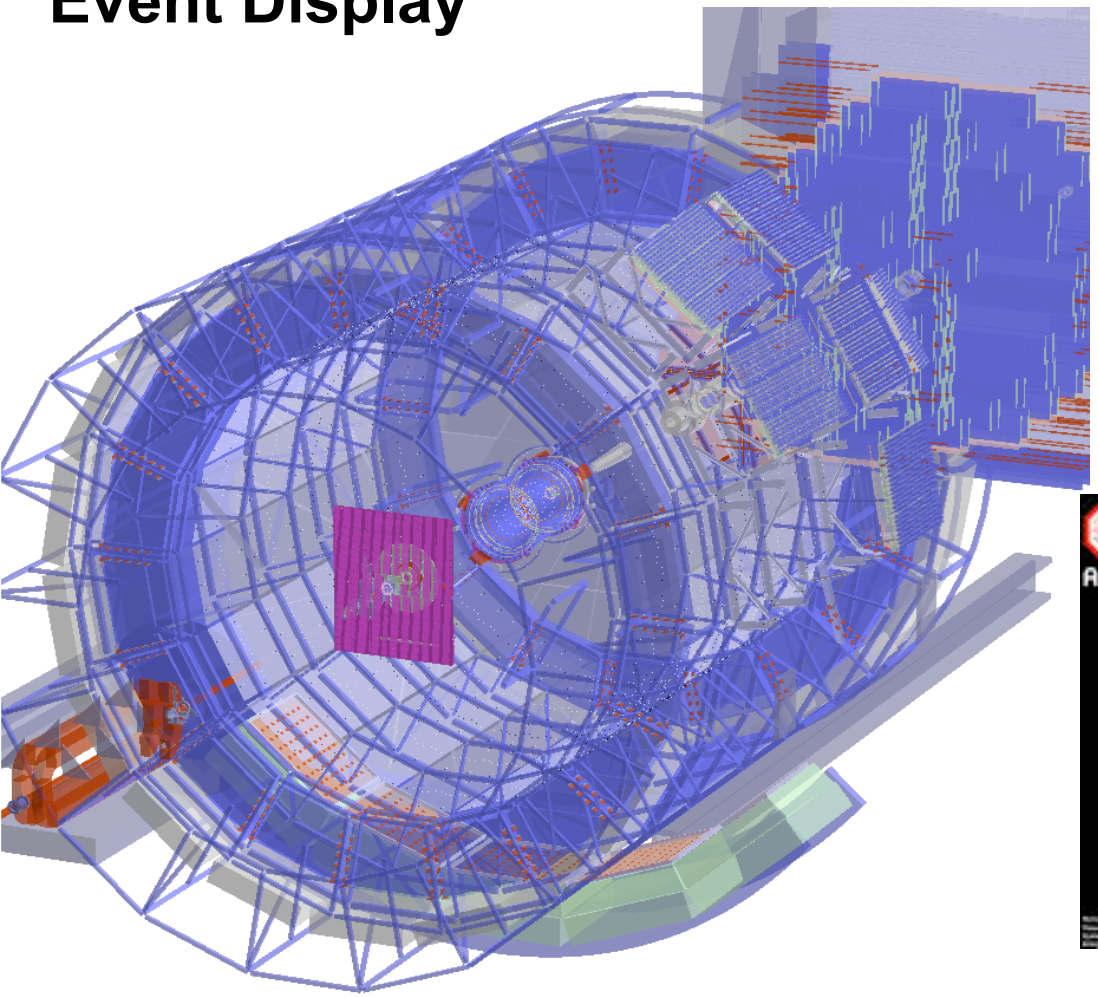


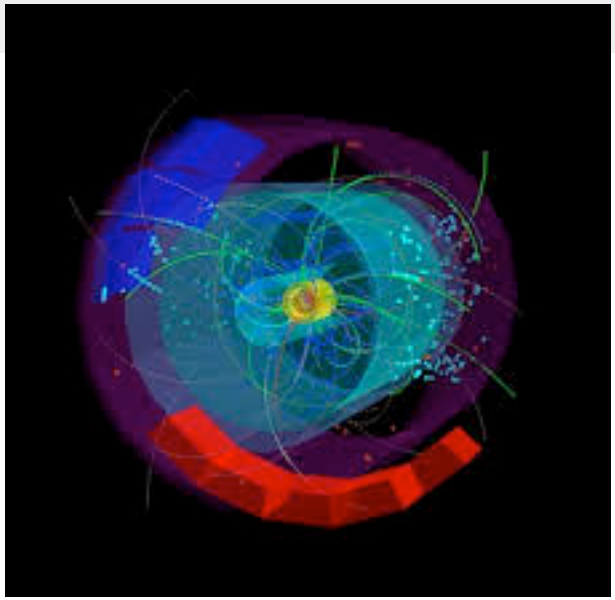


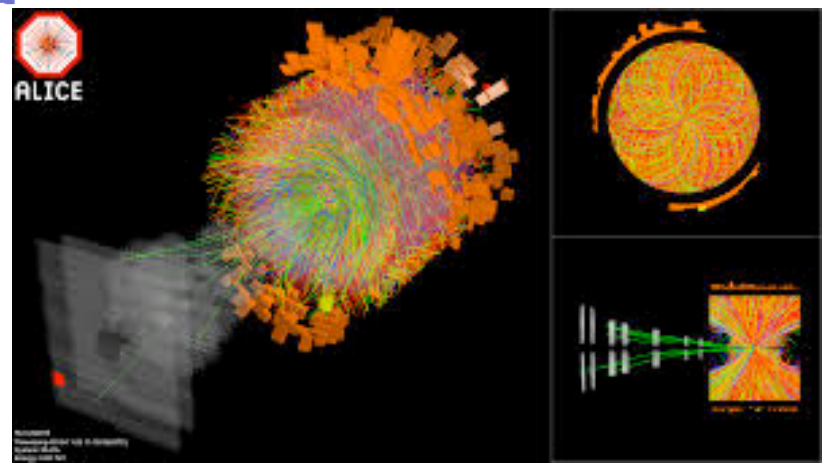


What is ROOT good for?

Geometry of ALICE detector + Event Display







ROOT as a project

- Open source project from the beginning
  - Available under GNU LGPL
- Started 1995 by R. Brun and F. Rademakers
  - Meanwhile ~10 full time developers
  - Many contributions from HEP experiments which use ROOT as base for their software frameworks
  - Strong feedback from the huge user community
    - Long list of small contributions
    - Bugfixes
    - …

ROOT

- ROOT is a software toolkit which provides building blocks for:
  - Data visualization
    - Thousands of plots in scientific publications and presentations
  - Data Analysis
    - Results of fits and parameter estimations
    - E.g. the Higgs Boson was discovered a ROOT based data analysis
  - Data processing
    - The software frameworks of the large LHC experiments are based on ROOT (Data reduction of the original data)
  - Data Storage
    - Hundreds of Petabyte of data is stored in ROOT files
- ROOT is mainly written in C++
  - Macro language is also C++
  - Python bindings available

ROOT

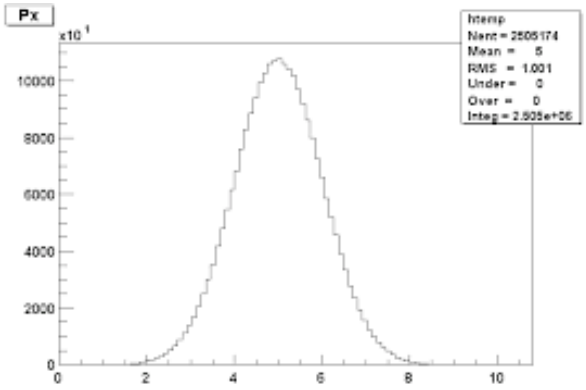
- ROOT is a toolkit (collection of building blocks) for various topics
  - C++ Interpreter
  - Graphics: 2D-histograms, 3D-histograms, graphs, …
  - Event Displays
  - Data Analysis: Trees, N-Tuple, Fitting, …
  - IO: row-wise ntuple storage, column-wise storage of C++ objects
  - Math: many mathematical objects
    - special mathematical functions (Bessel, Erf,…)
    - matrix
    - mathematical constants (π, …)
  - Statistical tools (RooFit/RooStats): rich modeling and statistical inference
  - Multivariate Analysis (TMVA): e.g. Boosted decision trees, neural networks
  - …

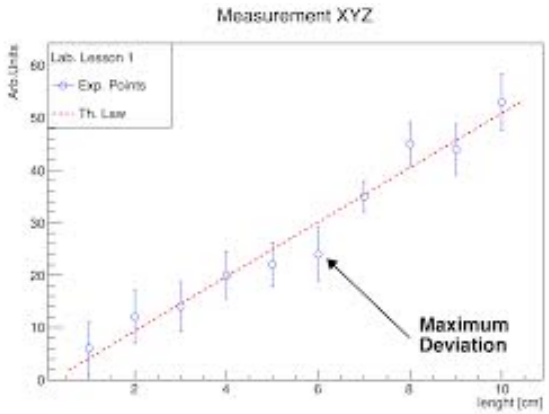
ROOT Interpreter

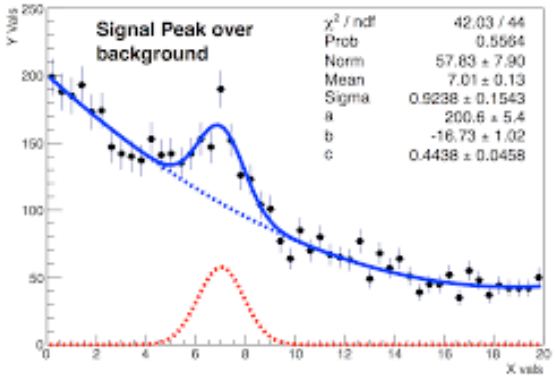
- In the moment there are two major versions of ROOT with different interpreters
  - ROOT 5: CINT (obsolete, don't use it any longer)
  - ROOT 6: CLING (will be used for the tutorial)

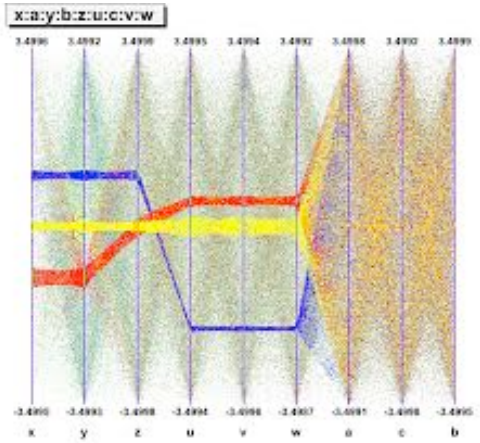
- CLING
  - C++ interpretation: highly non trivial and not foreseen by the language
  - Just In Time (JIT) compilation
  - A interactive C++ shell
  - Can interpret C++ code from a "macro" file
  - Allow rapid prototyping without explicit compilation cycle

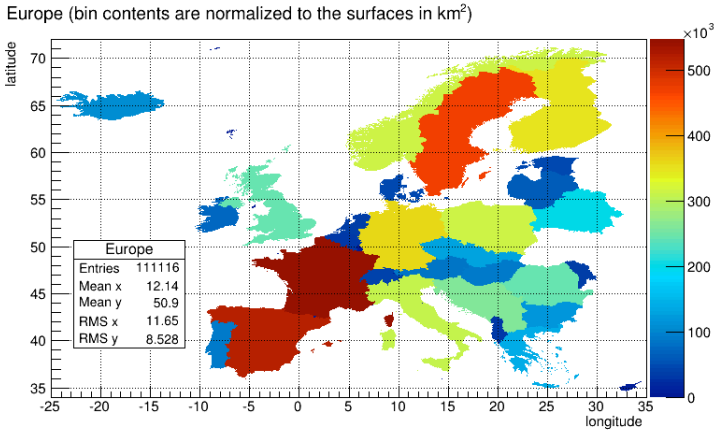
ROOT for visualization

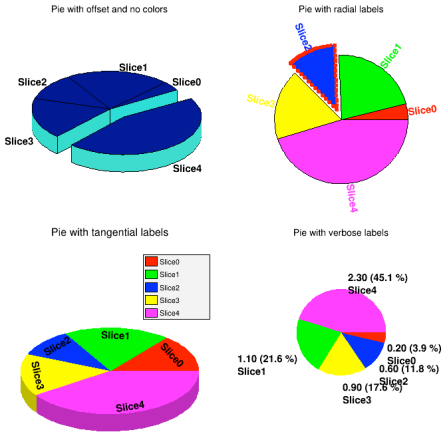






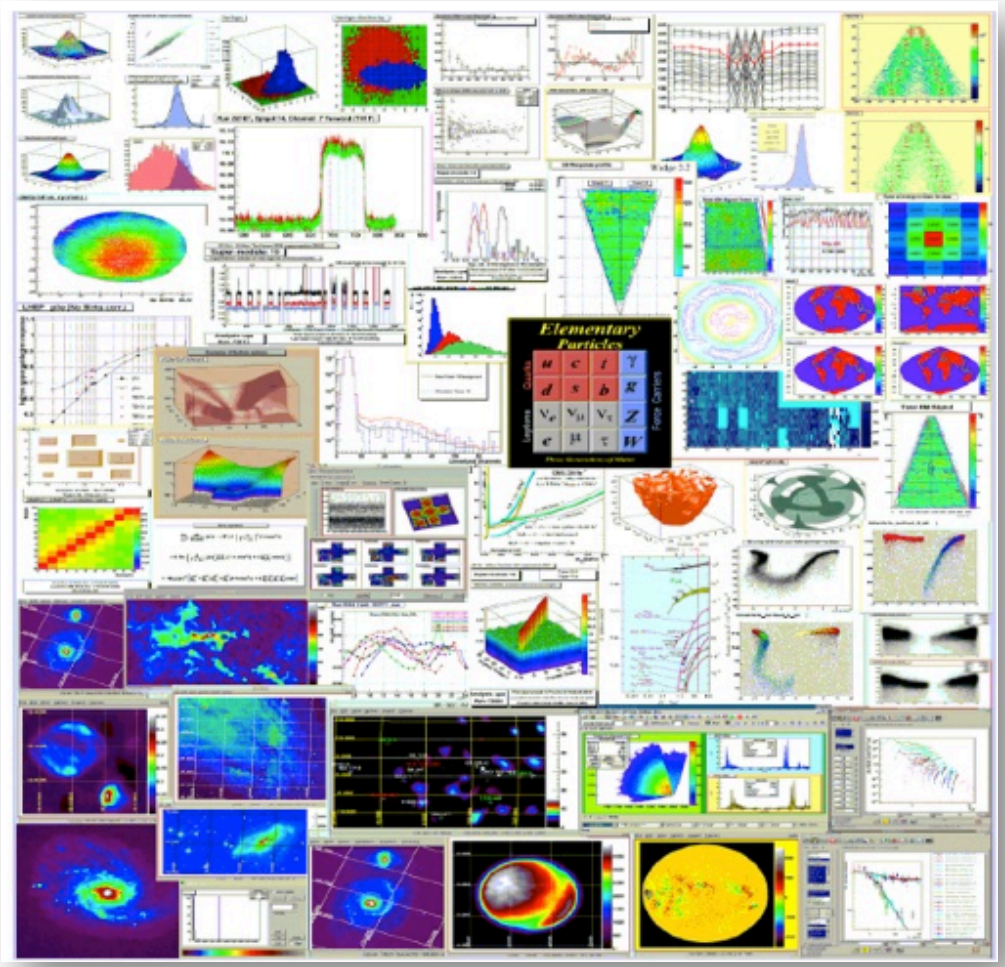




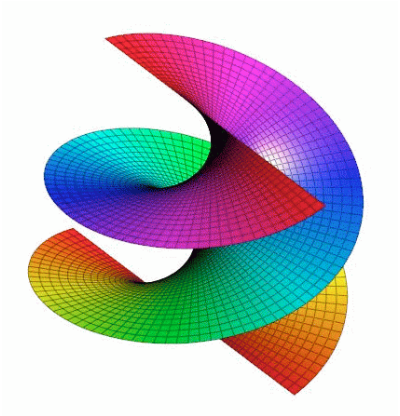


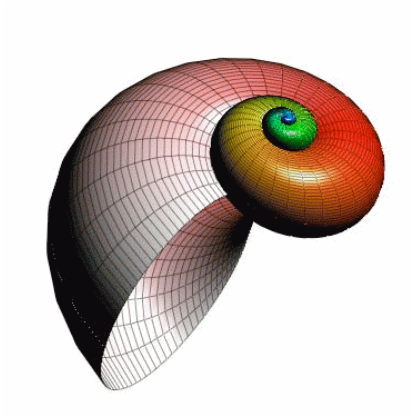
2D Graphics

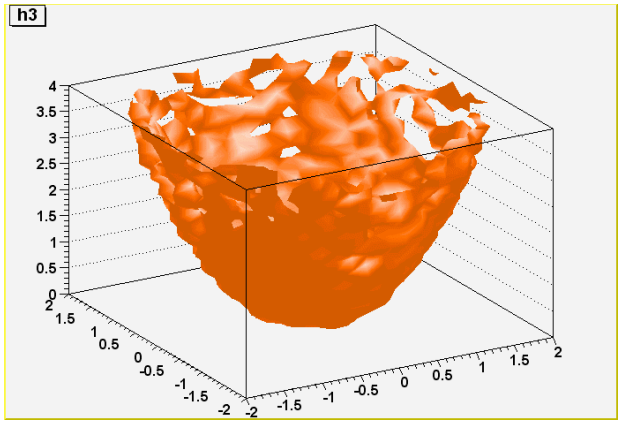
- Many different plot styles
- New ones added with each release
- Graphics can be saved in many formats
  - pdf
  - png
  - jpeg
  - Svg
  - ps
  - Latex
  - C

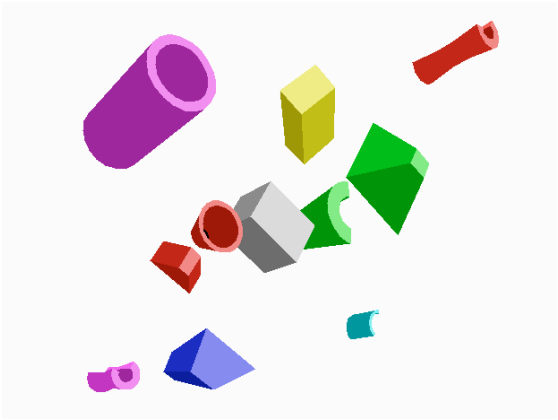


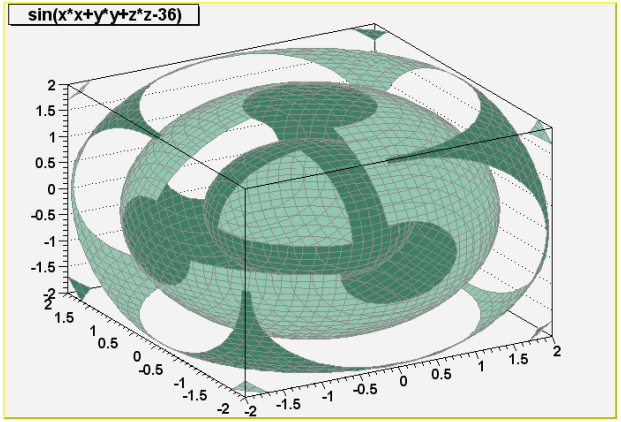
3D Graphics

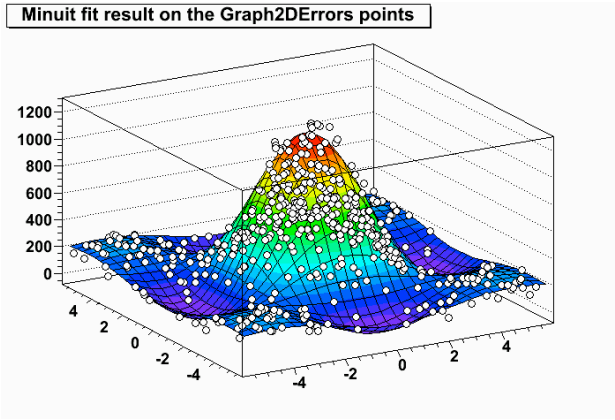





h3

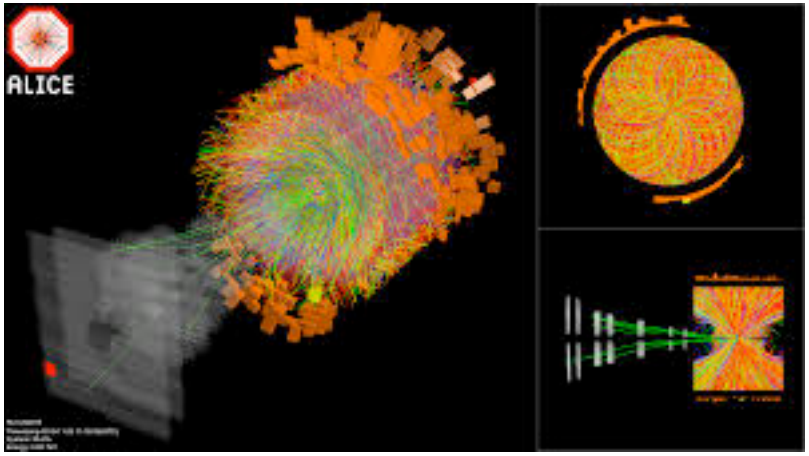


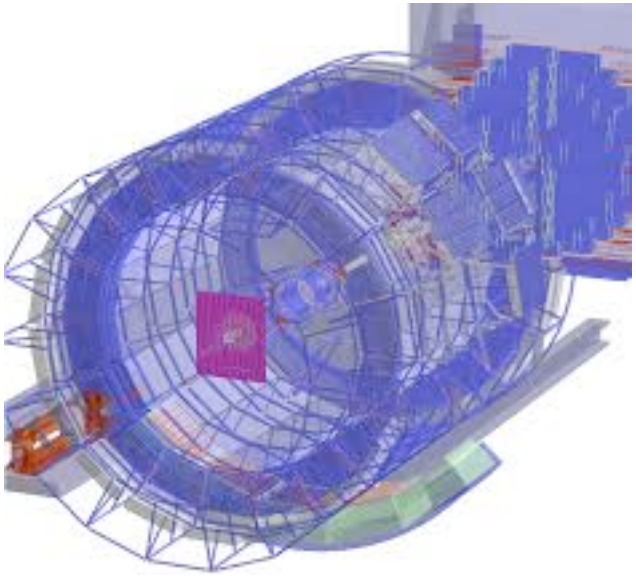

sin(x*x+y*y+z*z-36)

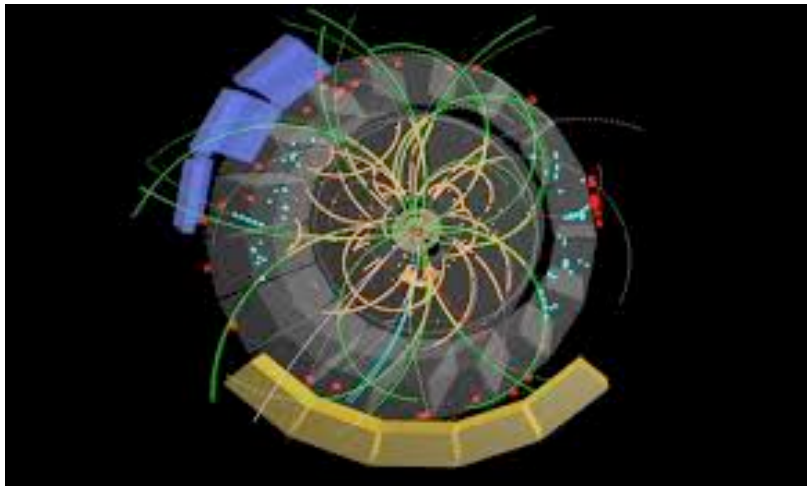

Minuit fit result on the Graph2DErrors points

Event Display and Geometry

- Geometry Toolkit
  - Build complex detector geometries
- Event Display (EVE)
  - Visualize particle collisions (events) within the detector geometries







ROOT for data analysis

- Simulation -> digitization -> reconstruction -> analysis
- Picture from CBM detector

ROOT IO and Persistency

- Data is stored in C++ objects
  - E.g. detector hit has a position, a timestamp and an energy loss
- How to save the information to a file?
  - Extract the data and create code for each data object
    - Not very convenient / error prone
  - Write/Read complex C++ objects to/from file
    - Called (de)serialization
    - Not possible with plain C++
    - Code to write/read to/from file is automatically created from the source code when compiling the source code
    - Used by HEP experiments to save Petabytes of data
    - IO Code is very performant
- Serialize the C++ objects using functionality provided by the C ++ interpreter
  - One function for ROOT based objects: TObject::Write()

A little bit of C++

- Compiled, strongly typed language
  - Compiler translates high level language into a format which is understood by the computer
  - Once compiled one only needs the produced executable
- Developed to get the best performance from the hardware
- Main computing language in HEP
  - High performance
  - Only one programing language to reduce costs for management of large code base
  - In the 90s most of the Fortran code used in HEP was migrated to C++

Fundamental C++ Data Types

- Character types (char, …)
  - Can represent a single character like “A” or “6”.
- Numerical integer types (short, int, long, …)
  - Can store a whole number value, such as 7 or 1024.
  - Exist in a variety of sizes, and can either be *signed* or *unsigned*, depending on whether they support negative values or not.
- Floating-point types (float, double, …)
  - Can represent real values, such as 3.14 or 15.8E-6, with different levels of precision, depending on which of the three floating-point types is used.
- Boolean type
  - The Boolean type, known in C++ as bool, can only represent one of two states, **true (1)** or **false (0)**.

Declaration Of Variables

- E.g:
  - int a;
  - bool c;
  - float myFloatNumber;
  - float a, b, c;
- Names for variables
  - Characters from a-z and A-Z, numbers, _
  - No special characters are allowed
  - “A” and “a” are different variables
  - No number as first character
  - Try to use meaningful names
    - E.g. “phone_number” instead of “xxx”
    - Help to understand the meaning of the code
  - Try to stick to a convention for the names

Initialization Of Variables

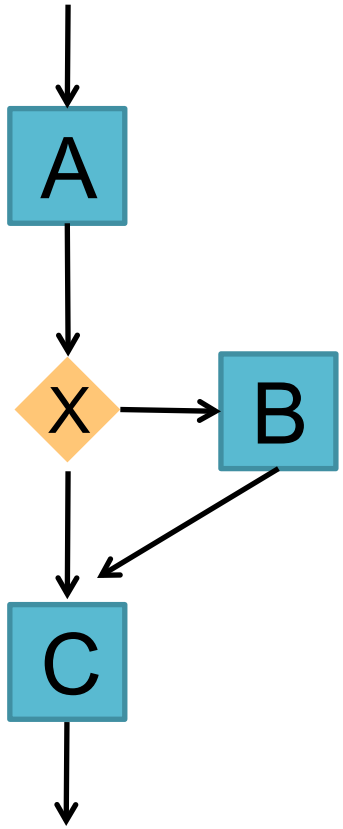
- Define a value when you declare the variable
  - int a = 5;
  - int a(5); // equivalent to “int a =5”
  - int a{5}; // equivalent to “int a =5”, since C++11
  - bool c = true;
  - float myFloatNumber = 1.23;
  - float a, b, c = 3.14158; // wrong
  - float a= 3.14158, b= 3.14158, c = 3.14158; //correct
- Try to always initialize a variable
  - Otherwise the value can be anything which depends on the compiler
  - Don’t assume 0 for integer and 0. for float
  - If you are using C++11 use the new form with curly braces

Statements

- Empty Statement
  - ;
- Simple Statements
  - Variable initialization
    - int a = 5;
  - Statements with operators
    - int y = a +10; // y = 15
    - E.g. arithmetic operators (+, -, *, /, %)
  - Compound statements
    - Statements grouped together in a block, enclosed in curly braces: {}
    - {int a=5; int y=a+10; … ;}
- Statements are executed in the same order in which they appear in the program
  - Always same result if not the input is different
    - Very boring
  - Need other statements to make programs more flexible

Selection Statement: if

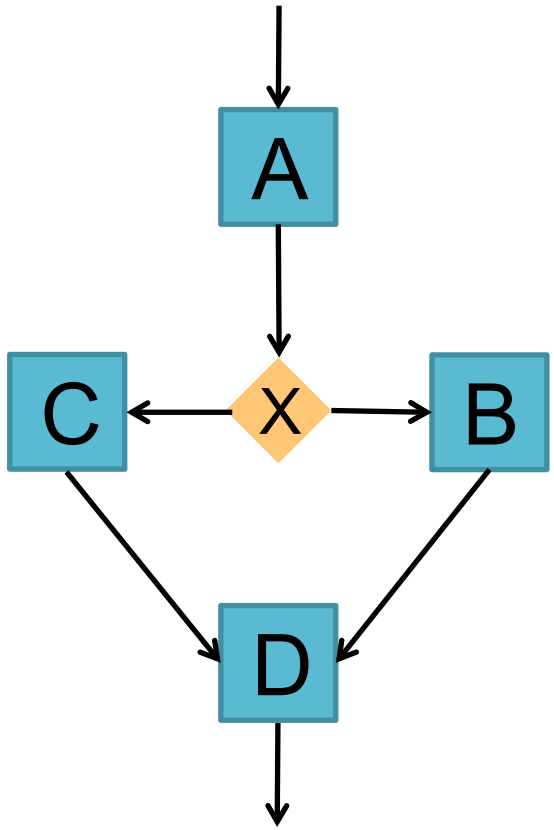
- if (condition X) statement B;
- Execute statement B only if condition X is true
- Execute several statements by grouping them into a compound statement {…}
- Important: In C++ every value beside 0 is evaluated as true



Selection Statement: if else

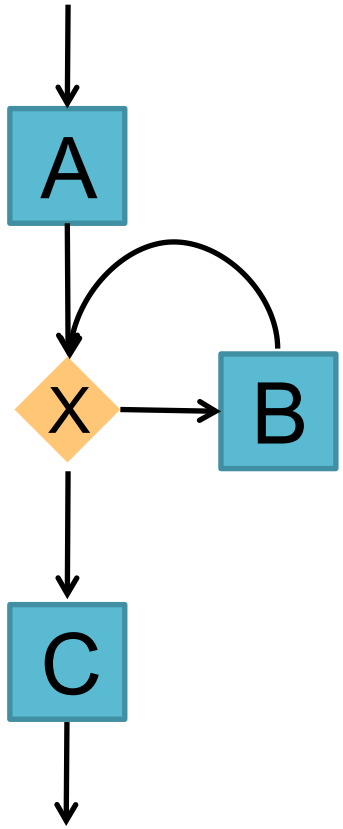
- if (condition X) statement B;
  else statement C;
- Execute statement B if condition X is true
- Execute statement C if condition X is false
- Use {} to group the statements

```
if (condition X) {
  statement B;
} else {
  statement C;
}
```



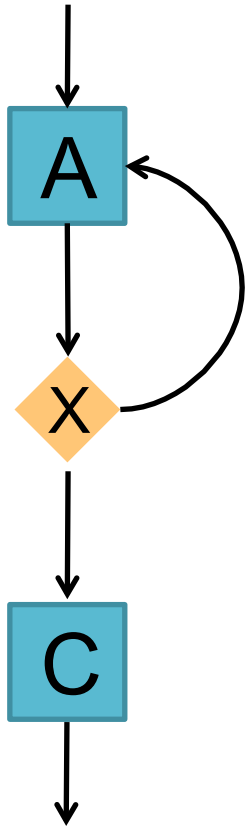
Iteration statements (loops) while loop

- while (expression X) statement B;
- The while-loop repeats statement B while expression X is true.
- It is possible that statement B is not executed at all (if expression X is false already the first time)
- If expression X never can become false in statement B you have created an endless loop



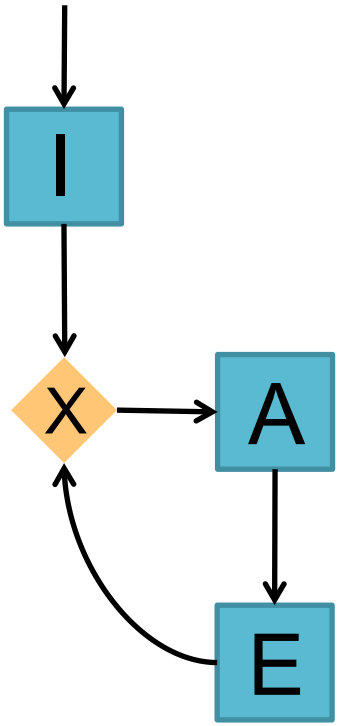
Iteration statements (loops) do while loop

- do statement A;
  while (expression X);
- Statement A is executed at least once, even if statement X is never true
- The do while-loop repeats statement A as long as expression X is true.
- If expression X never can become false in statement A you have created an endless loop



Iteration statements (loops)
for loop

- for (initialization I; condition X; statement E) statement A;
- Execute first statement I (Initialization)
  - Only executed once at the beginning of the loop
- As long as condition X is true execute first statement A and then statement E

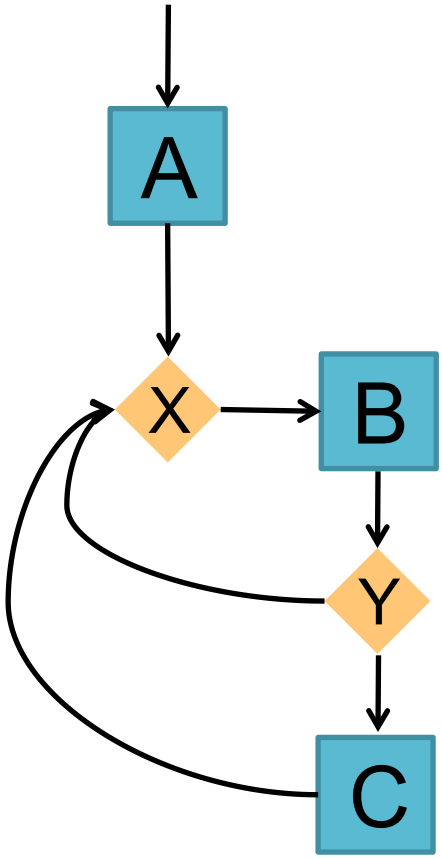



```
for (int counter = 0; counter < 10, counter++) {
   cout << counter << endl;
}
```

Jump statements continue

```
while (condition X) {
  statement B;
  if (condition Y) continue;
  statement C;
}
```

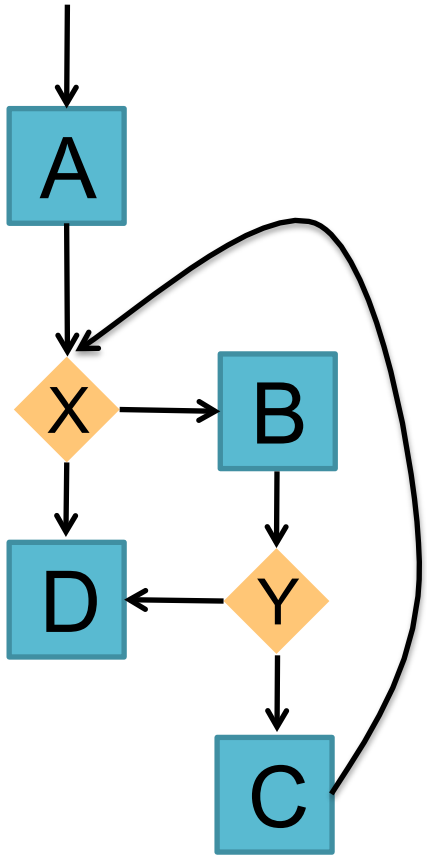
- Statement B is executed in each loop iteration
- If condition Y is true the loop jumps to the next iteration without executing statement C
- Statement C is only executed if condition Y is false



Jump statements break

```
while (condition X) {
  statement B;
  if (condition Y) break;
  statement C;
}
```

- Statement B is executed in each loop iteration
- If condition Y is true the loop jumps to the next statement after the loop (ending the loop
- Statement C is only executed if condition Y is false



C++ Class

- A “class” is an entity which encapsulates “data” and “actions” which can be performed on the data
  - The data is represented by the data members (variables of the class)
  - The actions are the class methods (functions of the class)
  - A method can have zero or more parameters
- An “object” is a concrete instance of the class
  - It is created by a special method the so called constructor
    - TH1F histo; // default constructor
    - TH1F* histo = new TH1F(“histName”, “HistTitle”, 64, 0, 64); // with params
- E.g. a histogram class has data points, a name, a title (data) and functions to plot or print this data

Objects And Pointers

- How to access methods and data of objects and pointers to objects?
- For objects use the dot operator “.”
- For pointers to objects use the arrow operator “->”
- Example:

TH1F histo (“histName”, “HistTitle”, 64, 0, 64);

```
cout << histo.GetName() << endl; // print histname
```

TH1F* histo = new TH1F(“histName”, “HistTitle”, 64, 0, 64);

```
cout << histo->GetName() << endl; // print histname
```

ROOT Basics

- Interactive ROOT session
- ROOT as a calculator
- Plotting a function using ROOT
- Plotting measurements
- Histograms

Interactive ROOT Session

- Normally C++ code has to be compiled
- ROOT offers a C++ interpreter
  - Use C++ interactively like Python, Ruby, Bash, …
  - Allows reflection (inspects class layout at runtime)
- Start with command "root" on the command line
- Special commands which are not C++ start with a "."
  - To quit root use .q
  - To execute a shell command use .!<command>
    - E.g. .!ls list the directory content
  - To get help use .help or .?
  - Load macros with .L <file_name>
    - More about macros will come later

Let's Start ROOT

- ROOT normally isn't installed to some system path
- Need to setup correct environment to execute "root"
- VirtualBox image
  - Login with user name/password combination: fairroot/FairRoot
  - Don't choose to upgrade to the newest Ubuntu version
  - Setup correct environment.
    - source /opt/root/bin/thisroot.sh
- GSI Linux cluster
  - Login with your user name and password
  - Login with default user (see information on table)
  - Setup correct environment
    - source setup_gsi.sh
- Your own computer
  - Depending on the directory setup the correct environment
    - source <your_root_install_dir>/bin/thisroot.sh
- Type "root" to start ROOT

ROOT as a Calculator

Now you should see something like this

```
demac019:~ uhlig$ root
   ------------------------------------------------------------
  | Welcome to ROOT 6.04/02                 http://root.cern.ch |
  |                               (c) 1995-2014, The ROOT Team |
  | Built for macosx64                                         |
  | From tag v6-04-02, 14 July 2015                            |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------------
```

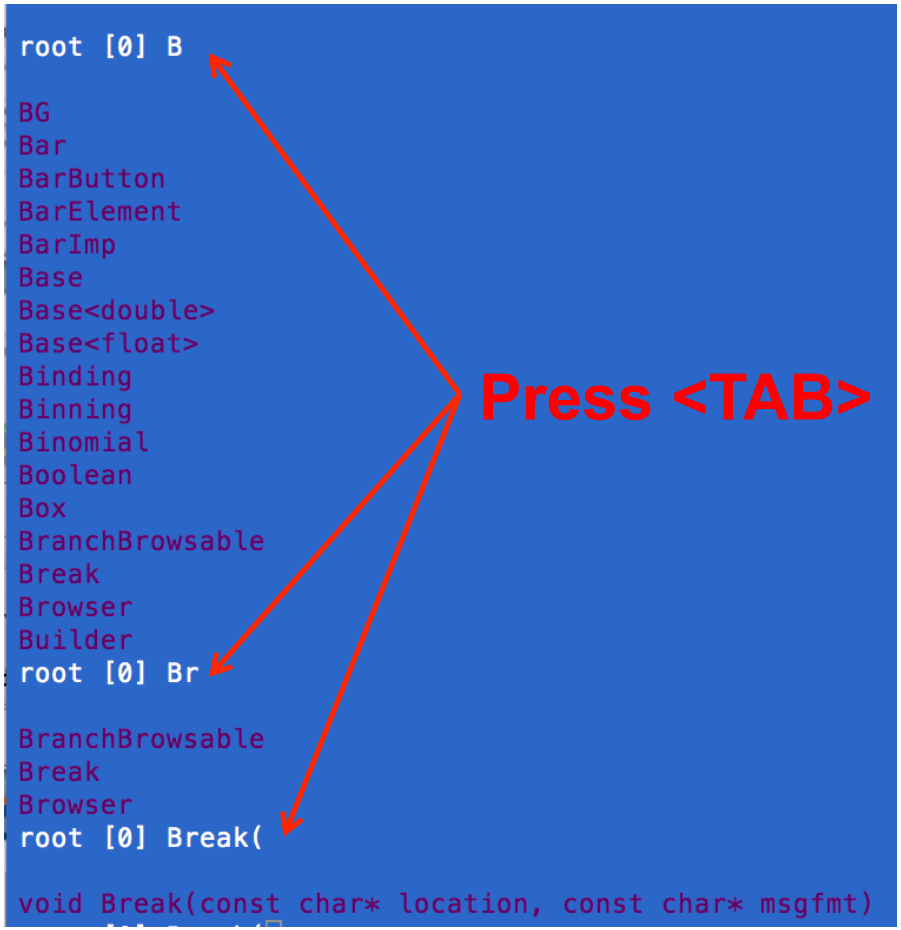
Do some simple calculations

```
root [0] 42+1
(int) 43
root [1] 2*(42+1)-33
(int) 53
root [2] sqrt(7.)
(double) 2.645751e+00
root [3] 1 > 3
(bool) false
root [4] 3 > 1
(bool) true
root [5]
```

Try it yourself

<TAB> is Your Friend

- If you don't know the command exactly type the first part and the press <TAB> which will show you all commands starting with what you have typed
- It even shows you some optional parameters



Exercise I

- Print the value of Pi
  - Hint: mathematical functions are in the namespace TMath::

Exercise I

- Print the value of Pi
  - Hint: mathematical functions are in the namespace TMath::
- Calculate the volume and the surface of a can with radius r=5cm and the height h=10cm

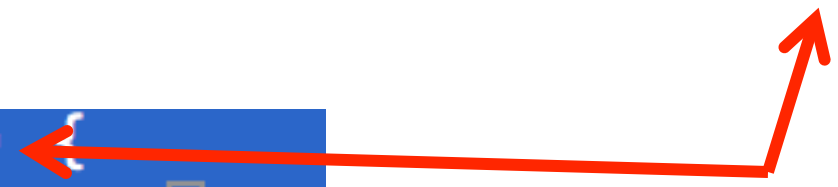
Exercise I

- Print the value of Pi
  - Hint: mathematical functions are in the namespace TMath::
- Calculate the volume of a can with radius r=5cm and the height h=10cm
- Calculate the volume of cans with the following combinations of radius and height
  - r=3. 5cm, h=8cm
  - R=25cm, h=3cm
  - R=10cm, h=9cm
  - R=9.76543cm, h=6.54378cm

ROOT Macros

- Up to now you have typed the lines of code interactively at the ROOT prompt
  - Not very convenient
  - What to do if there is a typo?
  - What to do if you want to rerun the same sequence of code?
- Use ROOT macros
  - Lightweight programs
  - Put there any code you have typed so far at the ROOT prompt
  - The general structure for a macro stored in file *MacroName.C* is

```
void MacroName() {
   < valid C++ code >
}
```



The name of the function and the file name has to be the same

Running A Macro

- Execute the macro from the command line

```
$ root MacroName.C
```

- Execute the macro from the ROOT prompt using ".x"

```
$ root
root [0] .x MacroName.C
```

- Load the macro into the ROOT session and execute it like that

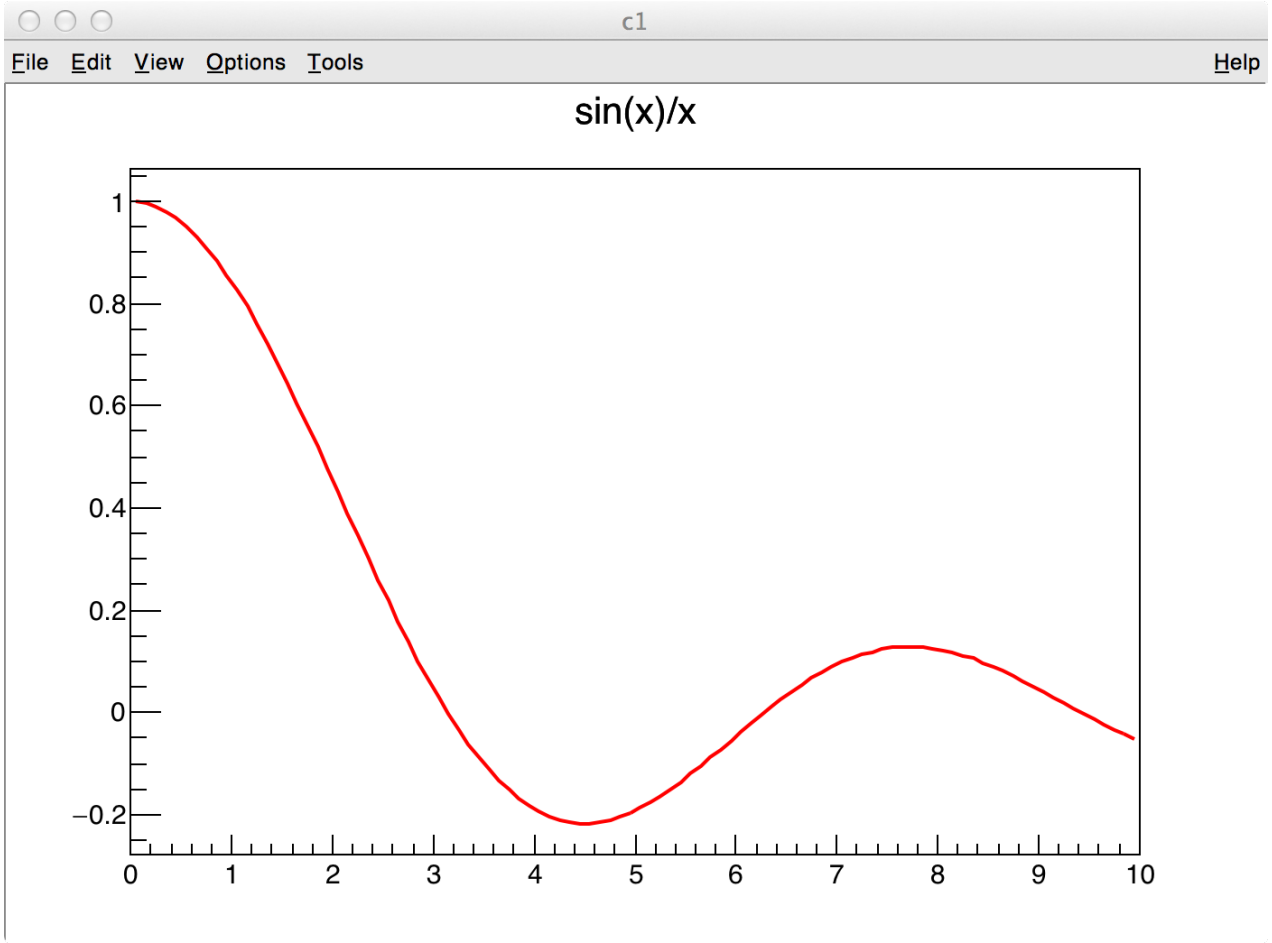
```
$root
root [0] .L MacroName.C
root [1] MacroName();
```

ROOT Data Types

- Basic data types (int, float, …) not clearly defined in the C++ standard
  - Not portable if you write them to file
- ROOT has it's own of data types which are portable
  - int -> Int_t
  - float -> Float_t
  - …
- In a program you can use either one
- If you want to write something to file you should use the ROOT types
- Better use them everywhere

Exercise II

- Try to plot the following function: y = sin(x)/x
  - Hint: Check the ROOT webpage at https://root.cern.ch

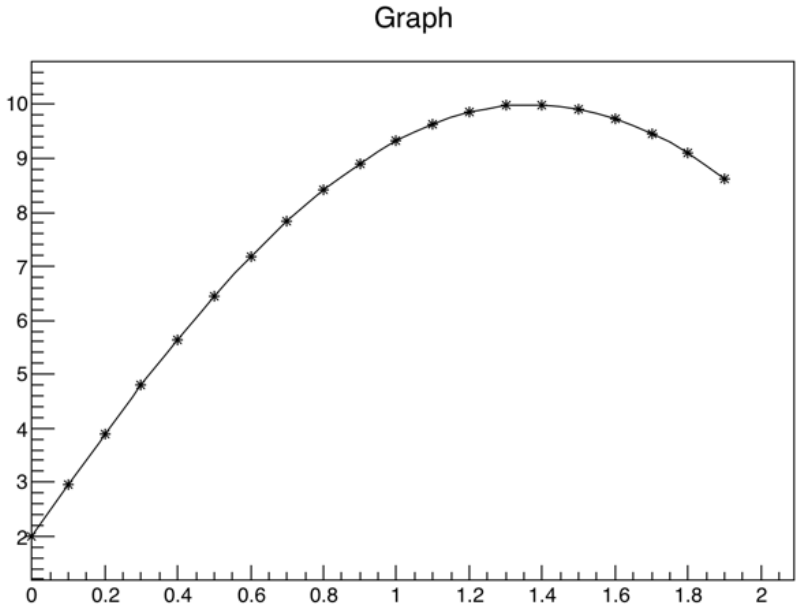


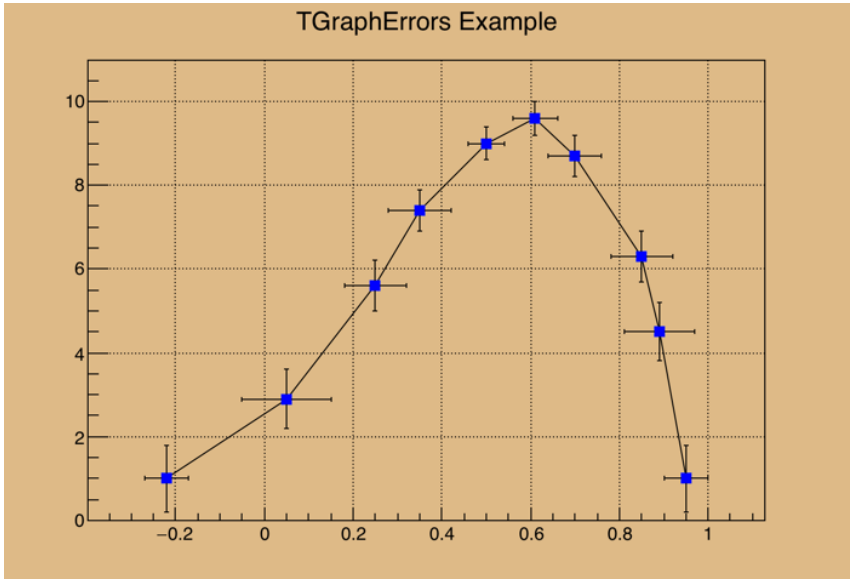
Exercise II

- Check what you can do with the histogram
  - Right-click on various parts of the histogram to open context sensitive menus
- Switch on ticks for the x- and y-axis
- Change the color and line type of the function to blue and dot-dashed
- Change the color of the histogram title to pink
- Save the canvas as root macro
- Exit ROOT
- Recreate the histogram from the macro

TGraph

- A TGraph is a graphics object made of two arrays X and Y with n points each.
- A TGraphErrors is a TGraph with error bars.
  - Arrays x, y x_error and y_error



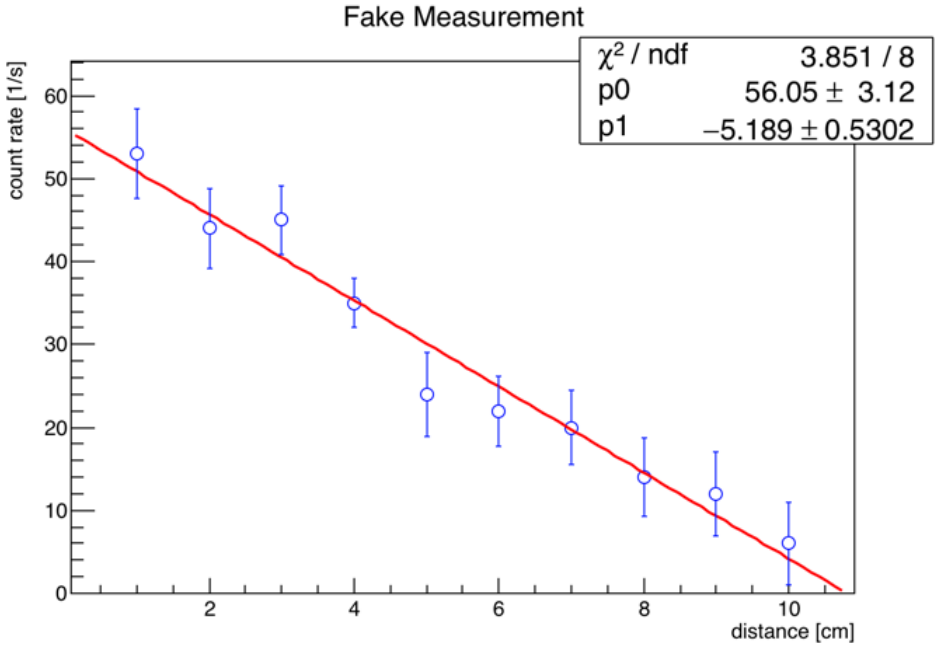


TGraph Exercise

- Draw a graph for the following points
  (x value, y value, y error)
  - (1., 53., 5.4),(2., 44., 4.8),(3., 45., 4.1),(4., 35., 2.9),(5., 24.,5.1),
    (6., 22., 4.2),(7., 20., 4.5),(8., 14., 4.7),(9., 12., 5.),(10., 6., 5.)
- Set the title to “Fake Measurement”
- Set the x-axis label to “distance [cm]”
- Set the y-axis label to “count rate [1/s]”
- Use blue open circles for the markers

Describe Data With A Model

- Use a linear function ($y=p_0+x*p_1$) to describe the data from the TGraph exercise and extract the parameters x_0 and x_1
  - Physicists call this action normally “fitting the data”
- Use the fit panel to select the function and start the fit



Histograms

- There are several classes for histograms
  - Name always starts with TH*
  - The third character defines the histogram dimension, e.g. TH1* for one-dimensional histograms (TH2*, TH3*)
  - The fourth character defines the maximum bin content of the histogram, e.g. TH1C
    - TH1C : one byte per channel. Maximum bin content = 127
    - TH1S : one short (2 byte) per channel. Maximum bin content = 32767
    - TH1I : one int (4 byte) per channel. Maximum bin content = 2147483647
    - TH1F : one float (4 byte) per channel. Maximum precision 7 digits
    - TH1D : one double (8 byte) per channel. Maximum precision 14 digits

Histograms

- Use appropriate histogram type for your problem
  - If histogram internal storage is to small you truncate the values at the maximum bin content (wrong results, no warning)

```
root [0] TH1C test("test","hist",100,0.,5.);
root [1] test.Fill(3., 200.);
root [2] test.GetBinContent(61)
(Double_t) 127.000
```

  - If histogram internal storage is to large you waste memory
    - Can become a problem if you use large (many bins) multidimensional histograms
    - TH3C with 1000 bins in each dimension has a size of 954MB
- Use the appropriate number of bins, lower and upper bounds for the histogram
  - If your values are integral numbers between 1 and 10 it doesn't make sense to have a histogram with 1000 bins from -10 to 90

TH* Exercise

- Create a 1-dim histogram with the proper number of bins, lower and upper bounds
- Fill the histogram with the sum of 3 dices
  - If you have 3 dices you can throw the dices, add the numbers and fill the value into the histogram
  - If you don't have 3 dices you have to simulate the experiment in the computer
    - TRandom is the ROOT class to draw random numbers
- After you have filled the first number repeat the throwing of the dices 10000 times
  - Maybe now it is time to do the experiment in software ;-}}

Input And Output

- How to save your results?
  - Up to now on only simple macros which can be easily started over and over again
  - What about programs which run for a long time?
- ROOT allows to write any C++ object derived from TObject to an output file
  - Something not supported by C++ itself
  - Simply use the function Write() of the object

```
void IoExample() {

   TH1F* histo = new TH1F("histo","title;X;Count rate",100,-5.,5.);
   histo->FillRandom("pol2");

   TFile* out_file = new TFile("outfile.root","RECREATE");
   histo->Write();
   out_file->Close();
}
```

Exercise

- Create the root macro IoExample.C copying the script from the previous slide
- Execute this macro
  - root IoExample.C
- Quit ROOT
- Read the created file back into ROOT
  - root outfile.root
- Open the browser
  - TBrowser b;
- Inspect the content of the file

Read A Object From File

- Reading is simple

```
void IoExample2() {

   TFile* file = new TFile("outfile.root","READ");

   file->ls();
   TH1* hist = NULL;
   file->GetObject("histo", hist);

   hist->Draw();
//   file->Close();
//   delete file;
}
```

Known Problems

- If you uncomment the commented lines you will only see an empty window
- Reason
  - The Object (in our case a Histogram) is “owned” by the TFile
  - The histogram is gone if you close the file
- C++
  - If an object (in our case TFile) is deleted or goes out of scope the destructor of this object is called
  - The destructor of The Tfile object deletes all objects which are “owned” by TFile
- To change this add TH1::AddDirectory(false) into the macro

Known problems

- Be careful when using pointers. They must be valid
- Check that you have a valid pointer before using it
- If not

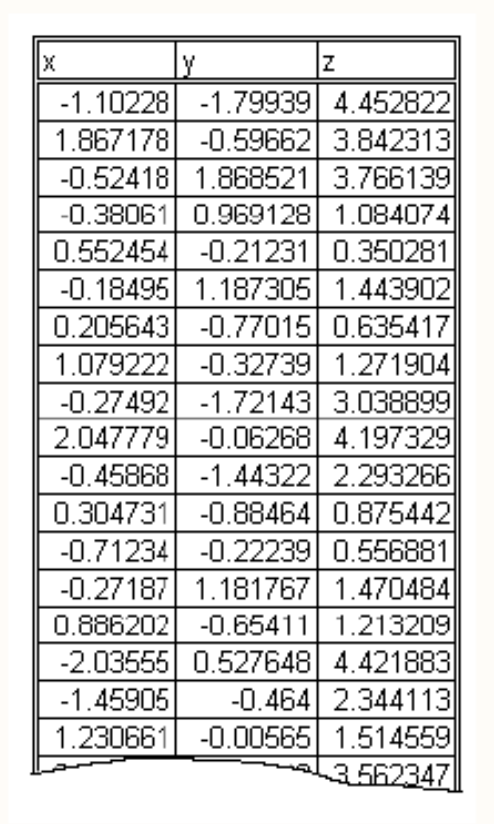
```
root [0] .x IoExample2.C
TFile**         outfile.root
 TFile*         outfile.root
  KEY: TH1F     histo;1 title

 *** Break *** segmentation violation
```

Better Code

```
void IoExample3() {

   TFile* file = new TFile("outfile.root","READ");

   if (file == NULL) {
     cout << "File not found." << endl;
     exit(1);
   }
   file->ls();

   TH1* hist = NULL;
   TString objName = "hist";
   file->GetObject(objName, hist);

   if (hist == NULL) {
     cout << "Object " << objName
          << " not found in file."
          << endl;
     exit(1);
   }
   hist->Draw();
   file->Close();
   delete file;
}
```

TNtuple

- A ROOT TNtuple object can store rows of float values
- It is like an Excel table with numbers
- Check the example TNtuple_write.C
- Run the macro to create the output file
- Open the file with ROOT
  - root -l conductivity_experiment.root
- Start the browser and examine the TNtuple
  - TBrowser b;
- What happens if you click on the leafs?



x	y	z
-1.10228	-1.79939	4.452822
1.867178	-0.59662	3.842313
-0.52418	1.868521	3.766139
-0.38061	0.969128	1.084074
0.552454	-0.21231	0.350281
-0.18495	1.187305	1.443902
0.205643	-0.77015	0.635417
1.079222	-0.32739	1.271904
-0.27492	-1.72143	3.038899
2.047779	-0.06268	4.197329
-0.45868	-1.44322	2.293266
0.304731	-0.88464	0.875442
-0.71234	-0.22239	0.556881
-0.27187	1.181767	1.470484
0.886202	-0.65411	1.213209
-2.03555	0.527648	4.421883
-1.45905	-0.464	2.344113
1.230661	-0.00565	1.514559
		3.562347

TNtuple Exercise

- Open the TreeViewer
- Within the TreeViewer
  - Draw the correlation between Potential and Current
  - Draw the correlation between Potential and Current with a condition on the temperature (T < 270)
  - Draw Current/Potential as function of the Temperature
- The examples above show how to navigate in the multidimensional parameter space and how to find correlations between the parameters
- Try to create the same plots from the command line
- Read back the data using the macro TNtuple_read.C

TTree

- More general case than TNtuple
  - Actually TNtuple is a special case of TTree
- TTree can store more complex types, e.g. objects
- Extremely efficient for write once, read many times
- Designed to store many events with the same data structure
- Access
  - Direct access to any event, any branch or any leaf even in the case of variable length structures
  - Best performance when reading the events sequentially
  - Designed to access only a subset of the object attributes (e.g. only particles' energy)
  - Makes same members consecutive, e.g. for object with position in X, Y, Z, and energy E, all X are consecutive, then come Y, then Z, then E. A lot higher zip efficiency!
- Only one event in memory
  - Only half true because of effective caching (read ahead)

Create, Fill And Write A Tree

- The most simple way to create a tree is very similar to TNtuple
  - Allows to save different variable types whereas TNtuple only store float values
- Please have a look at WriteTree1.C
- Create a branch for each of the variables to be stored
  - A lot of work for a large number of variables
  - Error prone

Create, Fill And Write A Tree

- A better way to create a TTree is to use a C++ class which defines all the variables
  - Only one place to do changes
- WriteTree2.C is a simple example of a particle transport simulation
- Check the file Gctrack.h
  - Define the variables needed for the particle transport
  - //! after the variable definition tells ROOT that this variable shouldn't be persistent (written to file)
  - There are more special comments which tell ROOT how to generate the streamer needed to write the object to file

Read A Tree From File