SUPA Graduate C++ Course

Lecture 4

S. Allwood-Spiers
University of Glasgow

Lecture 3 Recap

- Inheritance
- Polymorphism
  – Interfaces
- Templates
  – The Standard Template Library (STL)
    - Introduction
    - Complex Numbers
    - Vectors
    - Iterators
    - Algorithms
    - …Strings

Lecture 4 Overview

- Strings, Stringstream, Formatting
- Applications
  - CLHEP
    - Random
  - Root
    - Classes:
      - Histograms (TH1, TH1F, TH1D,TH2 etc..)
      - Ntuples (TNtuple)
      - Trees (TTree)
    - Exercises
      - Examples for you to compile and try
    - Techniques and tools:
      - TBrowser
      - Scanning (StartViewer)
      - Cuts
      - Root Macros (plotting)

Strings

- Provided via the string container
  - Includes inherited container member functions
  - Allows the use of iterators and algorithms

```
std::string str1("C++ was written by Bjarne Stroustrup");
std::string str2("Bjarne works for AT&T");
…
str2.push_back('.')
…
std::cout << "str1.length()= " << str1.length() <<std::endl;
```

- More functionality than cstring
- More obvious syntax

String Streams

- Can connect a stream to string: stringstream
  - Allows stream syntax to parse or create string

```
#include <string>
#include <sstream>
#include <iostream>

int main (){
   double d_value = 0.0;
   …
   std::string str1("7.045 ");
   std::istringstream inStr(str1);
   …
   // Collect a double value
   inStr >> d_value;
   …
   std::cout << "d_value=" << d_value << std::endl;
…
}
```

Extract from ex2/StringStream.cc

Stream Formatting

- Ex3 demonstrates some flags for formatting standard input and output
- Flags can be set with setf and unsetf

Format flag to be set

Field bitmask containing this flag

```
std::cout.setf(std::ios_base::scientific,std::ios_base::floatfield);
std::cout << "d_arr[5]=" << d_arr[5] << std::endl;
```

- Or directly:

```
std::cout << std::scientific << d_arr[5] << std::endl;
```

Working with other software

- Other software will often come in the form:
  - include/ directory: header files.
  - lib/ directory:
    - static libraries (compiled code that will be included in your executable) – “.a” suffix in linux
    - shared libraries (compiled code that will be loaded into your executable at run time) – “.so” suffix in linux
  - bin/ directory: executables.

Working with other software

- Compile time – Include path 'g++ -I‘

e.g.: g++ -c -I/path_to_clhep_build_area/include main.cc

- Link time – Library path 'g++ -L‘

e.g. g++ main.o -L/path_to_clhep_build_area/lib -lCLHEP -o MyProgram.exe

to link to the library libCLHEP.a

- Run time – LD_LIBRARY_PATH must be set to point to necessary shared object libraries.

CLHEP Introduction

- Provides basic classes for a range of particle and nuclear physics applications
  - e.g. 3 vectors and 4 vectors, geometry, random number generators.
  - Check if code has already been written
- This course was compiled with version 1.9.3.2

CLHEP Modules

- Units
- Vector
- Random
- RandomObjects
- Geometry
- Matrix
- Evaluator
- GenericFunctions

CLHEP Random

- Contains random number engines and generators
  - Engines provide the random numbers
    - Provided with documentation references in the header files.
    - Set the seed
  - Generators use input random number to produce another random distribution.
    - Generators have static `shoot` member functions.

Static member function: associated with the class, not a particular object.

Examples of CLHEP Random

```
#include "CLHEP/Random/RanluxEngine.h"
#include "CLHEP/Random/RandGauss.h"
#include "CLHEP/Random/RandExponential.h"

  long seed = 123456789;
  RanluxEngine randomEngine(seed,4);

  for(int i=0;i<10000;i++) {
    dat[0] = RandGauss::shoot(&randomEngine);
    dat[1] = RandExponential::shoot(&randomEngine);
  }
```

Extract from example5/Ntuples.cc

Root - Introduction

An Object-Oriented Data Analysis Framework

http://root.cern.ch

- Used by large particle physics experiments (also nuclear and astrophysics)
  - Histograms
  - Ntuples
  - Trees
- Graphical User Interface Libraries
- C++ Interpreted Environment via CINT
  - Can compile code or use interpreter

ROOT data types

- basic types: first letter is capitalised, end with suffix "_t":

  int → Int_t, float → Float_t, double → Double_t
- Names of root classes start with "T" e.g.
  - TDirectory, TFile, TTree, TH1F, TGraph, …
- Some ROOT types (classes):
  - TH1F - Histogram filled using floating precision data
  - TH1D - Histogram filled using double precision data
  - TFile – a file containing persistent data
  - TDirectory – a directory (useful to keep a TFile tidy/organised)
  - TTree – can store per-event info in branches and leaves
  - TF1 – 1-dimensional function, TF2, …
  - TString– a ROOT string object
  - TObjString – a persistable root string
- Excellent (clickable) documentation available for all of these:
  - http://root.cern.ch/root/html526/ClassIndex.html

Root I/O from C++

- `TFile`: Files can contain directories, histograms, trees. These are "persistent" objects.
- In ROOT, an object is made persistent by inheriting from `TObject`.
  - Calling the `Write()` member function of this class or a derived class causes:
    - Object to be written to the current directory
    - An associated key is defined according to the name supplied
- Open file, produce histograms, and call `Write`.

TFile Summary

- When a ROOT file is opened it becomes the current directory.
- Histograms, trees and ntuples are automatically saved in the file.
- When the file is closed the histogram and tree objects associated with the file are deleted.
- Any object derived from TObject can be written to a ROOT file. It has to be added explicitly.

Root I/O From C++

```
#include <TROOT.h>
#include <TFile.h>
#include <TH1.h>

int main(int argc, char *argv[]) {

  cout << "Opening root file: " << argv[1] << endl;
  TFile *rfile = new TFile(argv[1],"RECREATE","Histogram Example");
  if(rfile==0) {
    cout << "Could not create root file: " << argv[1] << endl;
    return 0;
  }
  ...histogramming...
  rfile->Write();
  rfile->Close();
  return 0;
}
```

Include needed header files

Extract from example4/Histograms.cc

Root Histograms

Open TFile

```
Int_t nbinsx = 100;
Axis_t xlow = 0.0;
Axis_t xup = M_PI;
TH1F *histo = new TH1F("histo","Sine Wave",nbinsx,xlow,xup);

Axis_t x;
Stat_t w;
for(int i=1;i<=100;i++) {
  x = M_PI/100.0*((double)i);
  w = sin((double)x);

  histo->Fill(x,w);
}
```

key: labels a memory location in file

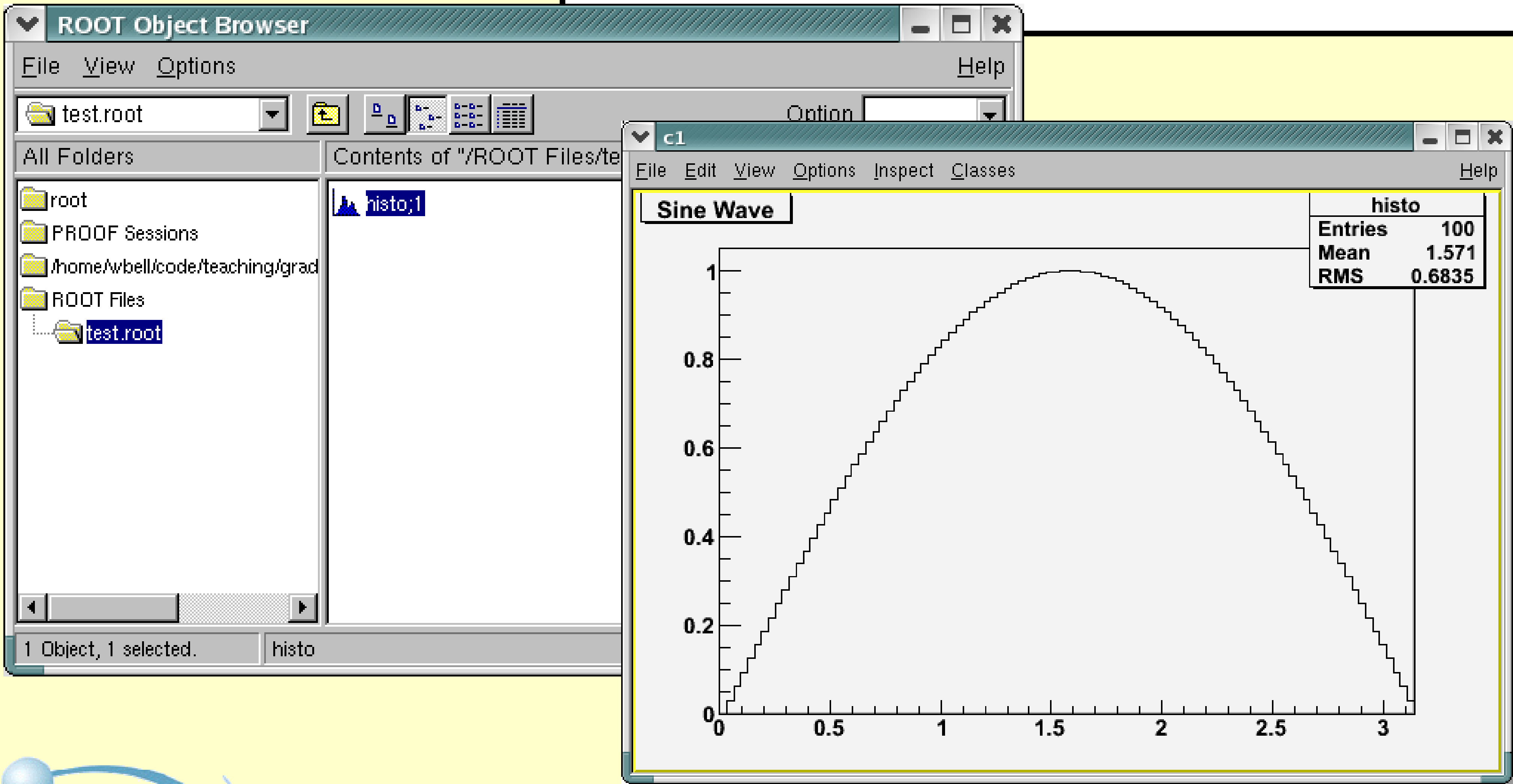
Warning: Root uses lots of typedefs for standard types.

Extract from example4/Histograms.cc

- Write and close TFile

Simple Plotting via CINT

```
[user@machine ex6]$ root test.root
root [0] new TBrowser();
```



Root `TNtuples`

- A simple TTree with TBranches of floats
  - Inherits from `TTree`
- Can be thought of as a table of data where each column is associated with a parameter.

TNtuple Example

- Open TFile

```
TNtuple *ntuple = new TNtuple("random_dat",
                              "Random Data","x:y:z");

Float_t dat[3];
for(int i=0;i<10000;i++) {
  dat[0] = RandGauss::shoot(&randomEngine);
  dat[1] = RandExponential::shoot(&randomEngine);
  dat[2] = dat[0]*dat[1];
  ntuple->Fill(dat);
}
```

Extract from example5/Ntuples.cc

- Write and close TFile

TTree

- TTree is a data structure of TBranches and TLeaves.
  – Each branch buffer can be individually accessed or accessed all together
  – Branches can be read from or written to different files.
- Ideal for large numbers of events
  – Allows compression of data

TTree Example

```
void writeTree(char *filename) {
  Float_t x, y, z;
  Int_t run, event;

  TFile *root_file = TFile::Open(filename,"RECREATE");
  if(!root_file) {
    std::cerr << "Error: could not open root file "
              << filename << std::endl;
  }
  else {
    TTree *tree = new TTree("tree","test tree");
    tree->Branch("Run",&run,"Run/I");
    tree->Branch("Event",&event,"Event/I");
    tree->Branch("x",&x,"x/F");
    tree->Branch("y",&y,"y/F");
    tree->Branch("z",&z,"z/F");
    ...
```

Extract from example6/Trees.cc

Branch name, address, leaf name and type

TTree Example

```
TRandom r;
for (Int_t i=0;i<10000;i++) {
  if (i < 5000) {
      run = 1;
  }
  else {
      run = 2;
  }
  event = i;
  x = r.Gaus(10,1);
  y = r.Gaus(20,2);
  z = r.Landau(2,1);
  tree->Fill();
}

tree->Print();
root_file->Write();
delete root_file;
```

Extract from example6/Trees.cc

TTree Example

```
  TFile *root_file = TFile::Open(filename,"READ");
  ...
    TTree *tree = (TTree*)root_file->Get("tree");
    Int_t entries = tree->GetEntries();
    TBranch *run_branch = tree->GetBranch("Run");
    ...
      Float_t x, y, z;
      Int_t run, event;
      ...
      run_branch->SetAddress(&run);
      ...
      for (Int_t i=0;i<10;i++) {
        tree->GetEvent(i);
        std::cout << event << " " << run << " "
                             << x << " " << y << " " << z <<
std::endl;
      }
```

Set the address to which objects from run_branch will be loaded

Data are written to addresses

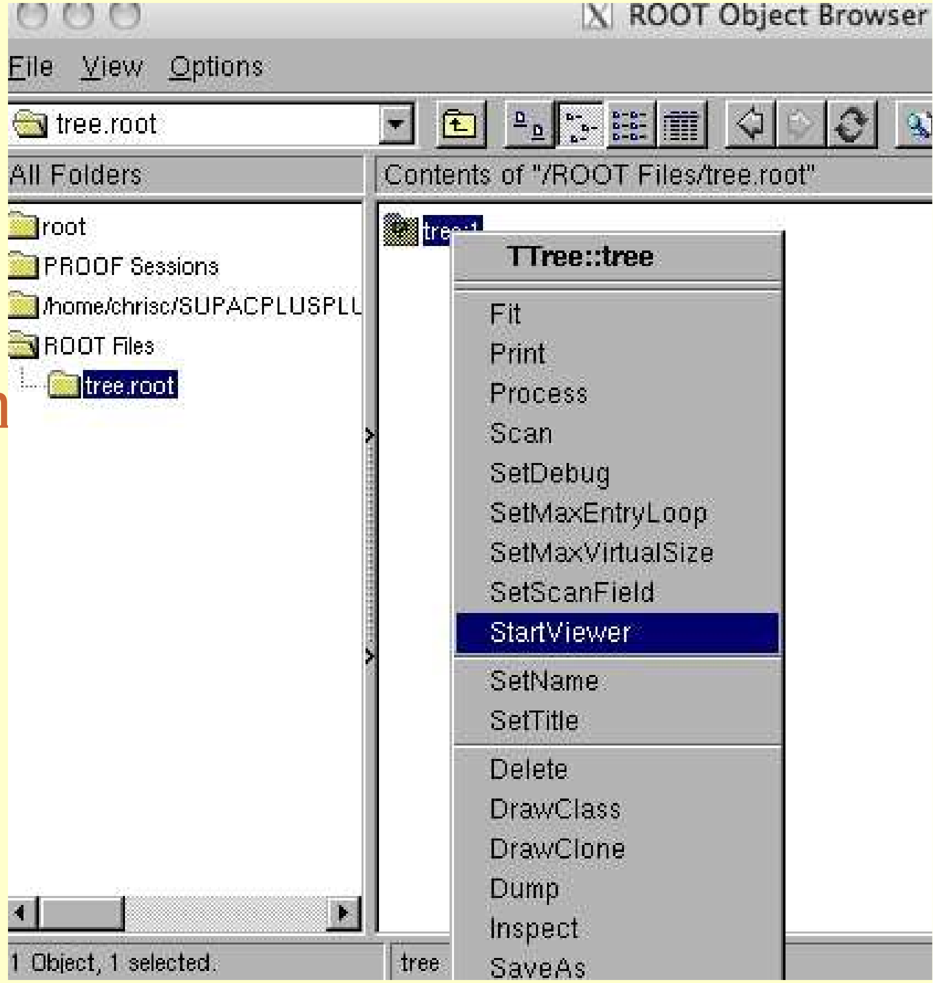
Extract from example6/Trees.cc

Exercises

- Session 4 examples:
  - Download examples and course guide from My.SUPA
    - You can download and build ROOT and CLHEP.
    - After untarring: tar –xvf gradcpp_lecture4_material_2011.tar
    - Change directory: cd gradcpp_lecture4_material
    - Examine README file:
      - Compile CLHEP, (ROOT-only if needed), and set up PATHS..
      - Compile CLHEP e.g. type: cd CLHEP; ./build-clhep-2.sh; cd ..
      - Then type: cd EXAMPLES-AND-PROBLEMS
      - Setup paths: source CLHEPsetup.sh; (source ROOTsetup.sh;)
  - Build and test examples.
    - Each example has its own README file

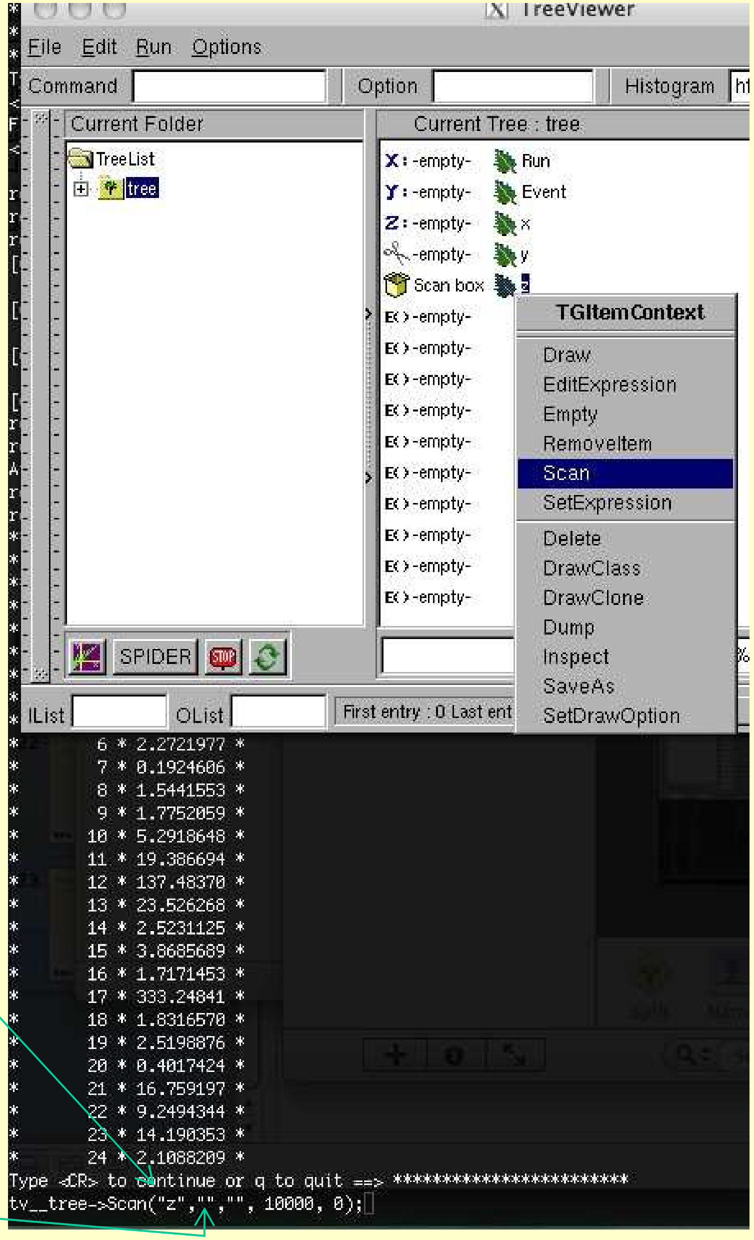
Techniques and Tools: Scan/StartViewer I

- We already saw TBrowser().
  - Now can use it to examine "tree.root" from example 3.
  - Instead of just clicking on 'tree' (the folder with the green tree in it) we can *right-click*:
    - StartViewer



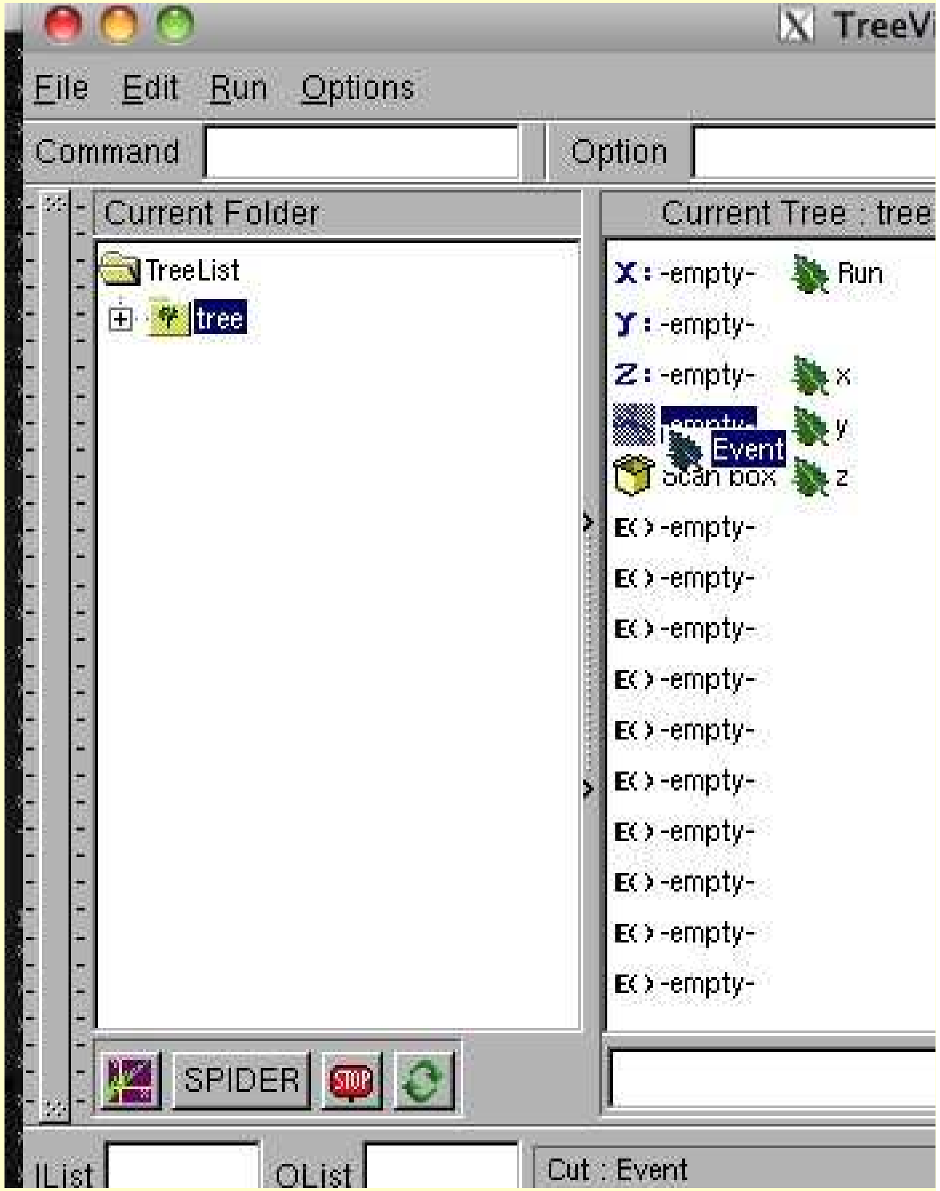
Techniques and Tools: Scan/StartViewer II

- StartViewer starts a new, more powerful TreeViewer window.
- Possible now to scan the values
  - right-click, select Scan.
- Can also *up-arrow* ↑ to recall the 'scan' command!
  - You can modify this manually- very powerful.
  - Colons to separate variables e.g. z:x
  - Cuts can be added to second set of "" e.g. "x>10.0"



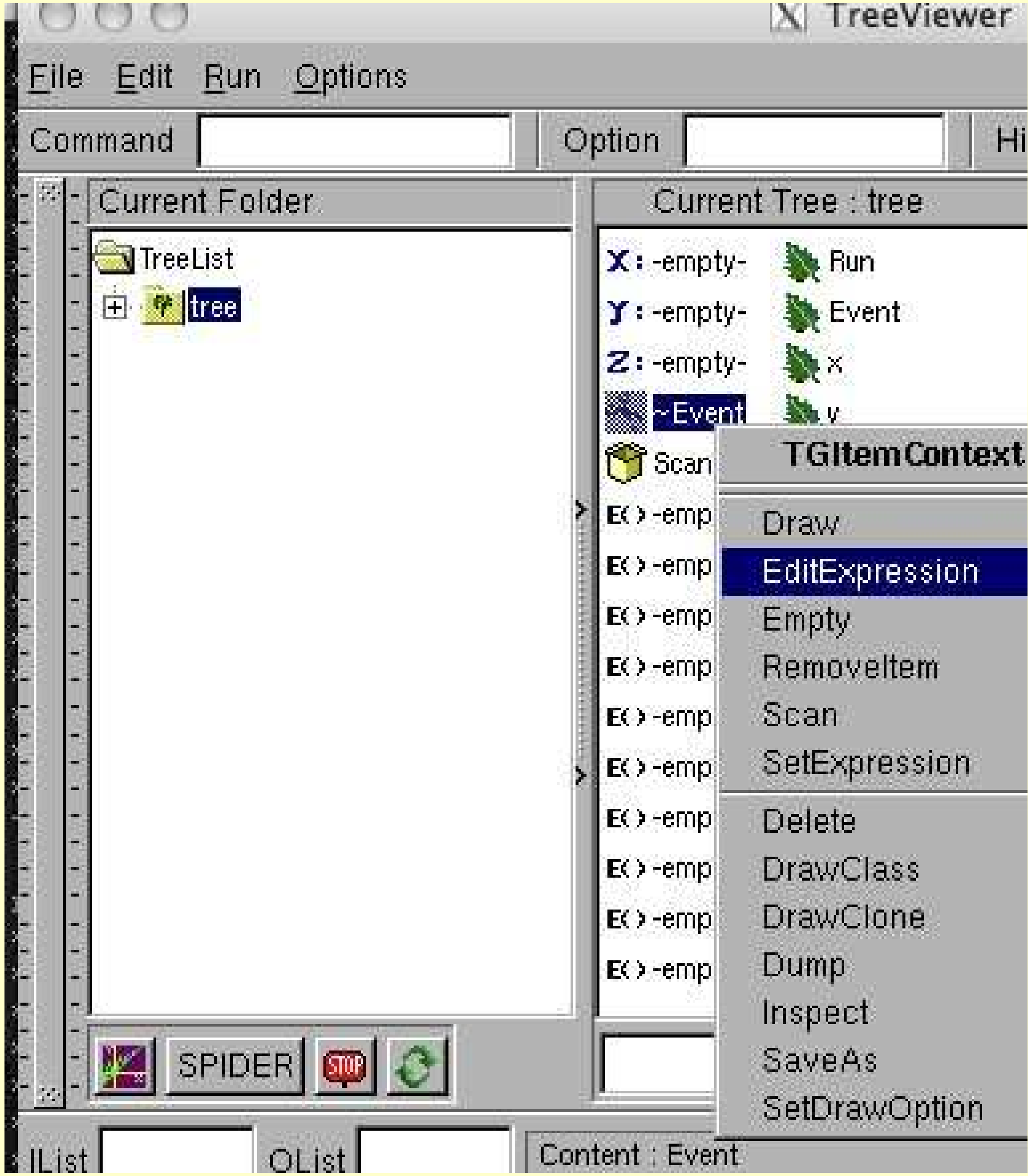
Techniques and Tools: Cuts/StartViewer I

- You can apply cuts in the TreeViewer window:
  - Drag a leaf e.g. Event over the scissors.
  - Drop it.



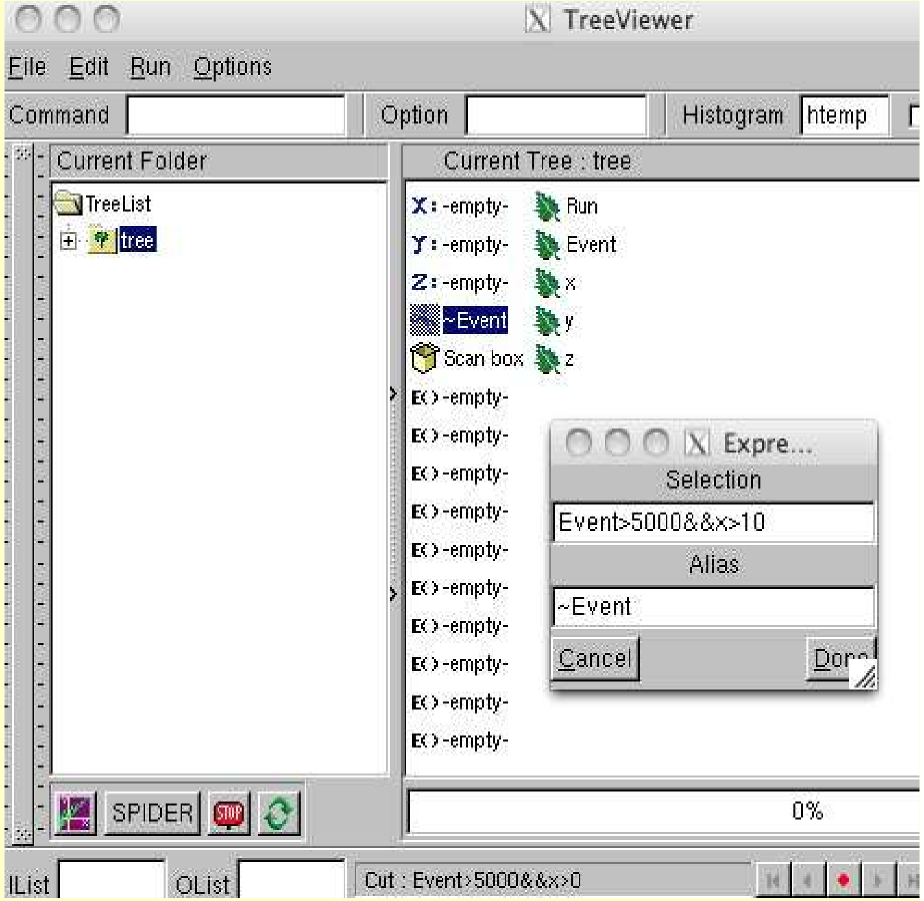
Techniques and Tools: Cuts/StartViewer II

- You can apply cuts in the TreeViewer window:
  - Drag a leaf e.g. Event over the scissors.
  - Drop it.
  - Right-click, select "EditExpression"



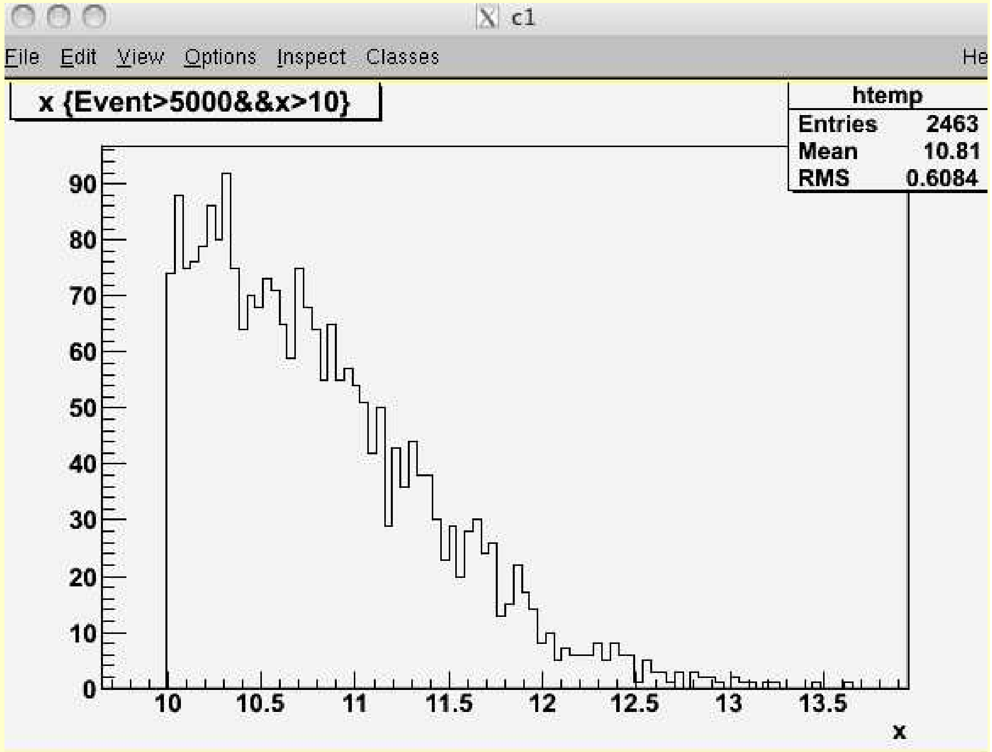
Techniques and Tools: Cuts/StartViewer III

- You can apply cuts in the TreeViewer window:
  - Drag a leaf e.g. Event over the scissors.
  - Drop it.
  - Right-click, select “EditExpression”
  - Insert a c-like expression using the variables in the tree
  - Click ‘Done’



Techniques and Tools: Cuts/StartViewer III

- You can apply cuts in the TreeViewer window:
  - Drag a leaf e.g. Event over the scissors.
  - Drop it.
  - Right-click, select “EditExpression”
  - Insert a c-like expression using the variables in the tree
  - Click ‘Done’



- Click on a variable to plot it..
- Here, I clicked ‘x’
- ‘x’ appears with the cuts applied

Root Macros

- Typically used to make plots repeatedly.
- Use { and } at the start/end of the macro.
- Typical macro: mymacro.C from example 3.
  - First, run “./Trees.root –w” to create the output events.
  - At the root command line, type .x mymacro.C

```
{…
//open the file and get the TTree called 'tree'
TFile myfile("tree.root");
TTree* mytree= (TTree*)gDirectory->Get("tree");
//create a canvas
workingcanvas=new TCanvas("workingcanvas","",0,0,600,400);
//set up histograms and draw them on the canvas
xGreaterThan10= new TH1F("xGreaterThan10","Plot of x",50,5,15);
workingcanvas.cd(1);
mytree->Draw("x>>xGreaterThan10","x>10");
```

Extract from example3/mymacro.C

That's all folks!

Well, nearly…

- Tutorial (optional): Monday 3rd December 11am-1pm, room 320 Kelvin Building.
- Section 2, problem 2 and Section 3 problems 1 and 3: deadline Tuesday 4th December.