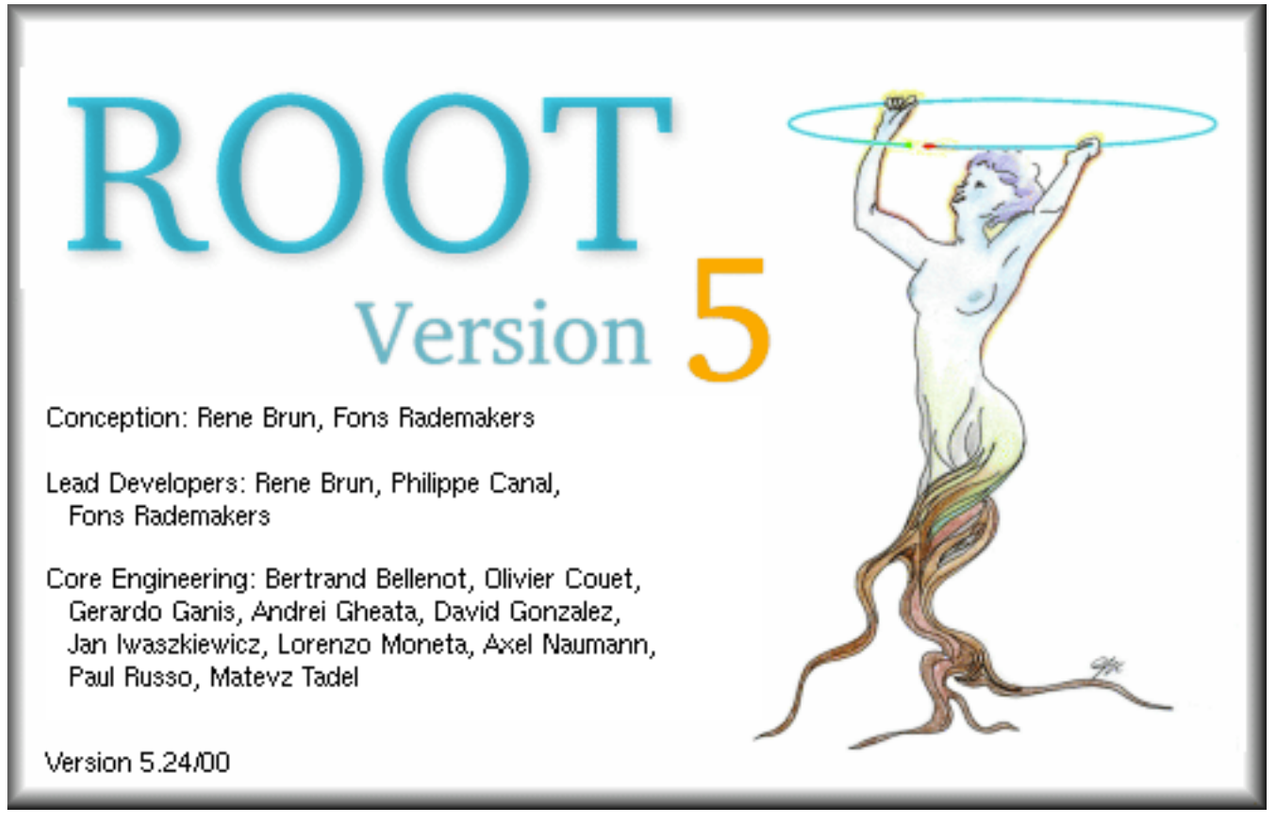



# Lecture I: Basics

Linux commands
What is ROOT?
Interactive ROOT session
  - command line vs. macros vs. user-compiled code
Opening files / accessing information
Histograms and Trees and Functions, Oh My!

# Some basic linux commands

| Linux command | What it does... |
|---|---|
| `ls` | List contents of a directory |
| `pwd` | Show present working directory |
| `mkdir test` | Make a new directory called test |
| `cd test` | Change to directory test |
| `cp file1.txt file2.txt` | Copy file1.txt to file2.txt |
| `mv file1.txt file3.txt` | Move file1.txt to file3.txt |
| `cat file2.txt`<br>`less file2.txt`<br>`more file2.txt` | Print the contents of a file to the screen |
| `emacs -nw`<br>`vi`<br>`pico`<br>`...` | Console text editors (no extra window pops up) |
| `emacs`<br>`xemacs`<br>`nedit`<br>`gedit`<br>`...` | GUI text editors (extra window pops up) |

# What is ROOT?

Versatile software package developed for performing data analysis

- Read data from some source
- Write data to a file
- Select data with some criteria (“cuts”)
- Perform calculations and fits
- Produce results as plots, graphs, numbers, fits, etc.
- Save results in some format (ROOT file, image of plot, ...)

# ROOT can be used in many ways:

## Command line

Good for quickly making plots, checking file contents, etc.

## Unnamed macros

Execute commands as if you typed them on the command line

List of commands is enclosed by one set of { }.

Execute list of commands from command line by: “`.x file.C`” (without quotes)

## Named macros / Compiled code

Best for analysis, can be compiled and run outside of ROOT, or loaded and executed during interactive session

# Developed and supported by high energy physics community

Homepage with documentation and tutorials (http://root.cern.ch)

Google will also find the documentation (for example, try “root TH1F”)

# Syntax

Many of the commands we will use will have this general form:

This is called a "constructor."

```
TSomething* mything = new TSomething(stuff);
```

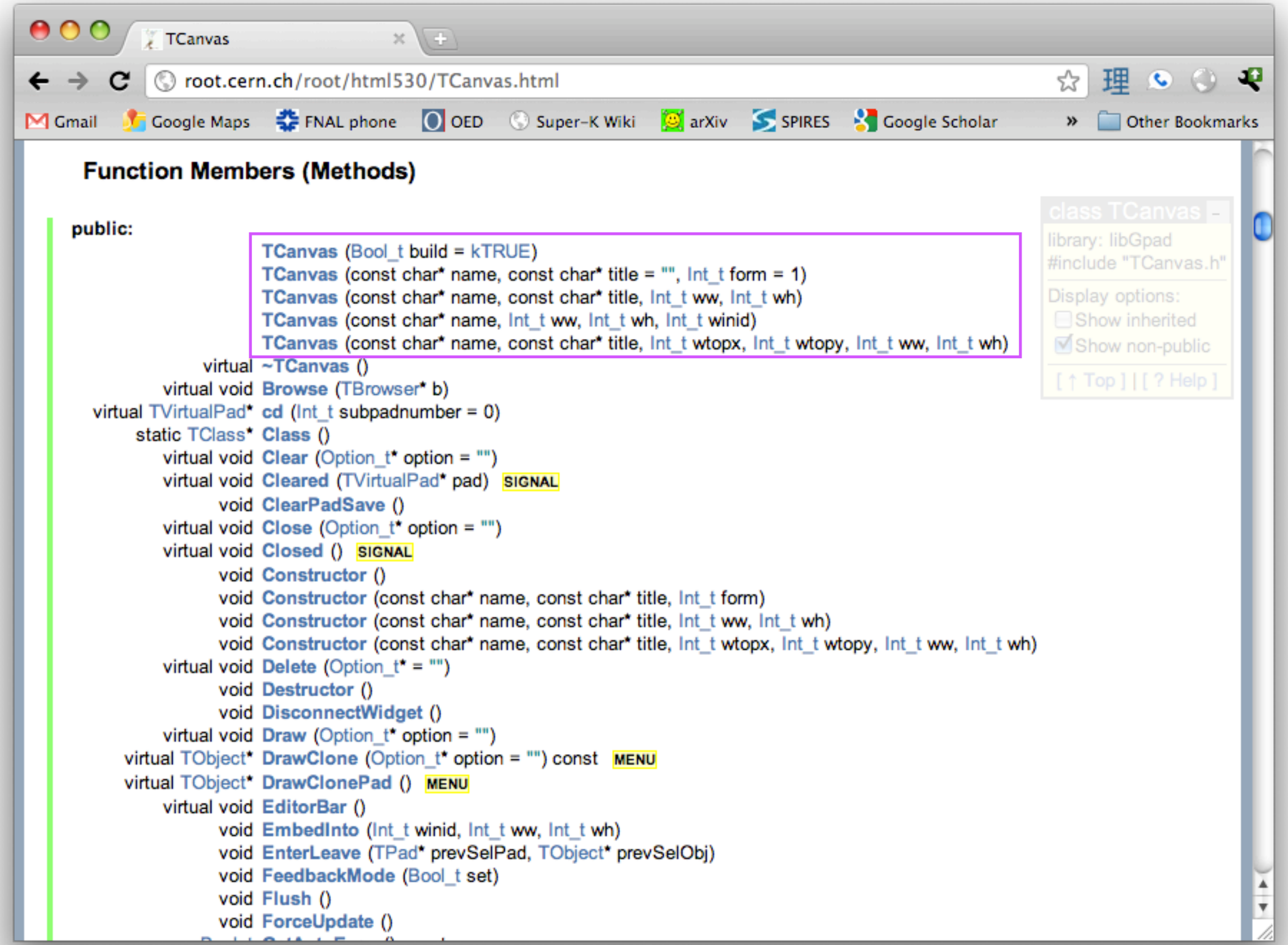
- `TSomething` — All ROOT classes start with T:
  - TFile
  - TH1
  - TTree
  - TCanvas
  - ...
- `*` — Means make a "pointer" (in this case, of type TSomething)

  For now, don't worry about what a pointer is. It's not important for this tutorial.
- `mything` — Name of pointer
- `new` — C++ operator that allocates memory
- `TSomething(stuff)` — Initializes the allocated memory with whatever "stuff" TSomething requires.

  Look at ROOT class definition to find out what "stuff" you must provide.

Note: In C++, if you allocate memory using the "new" operator, you must later use "delete mything" to release the memory... otherwise your code will have a memory leak. We will not worry about that today, but keep it in mind for your future code-writing.

# Example: TCanvas Constructors



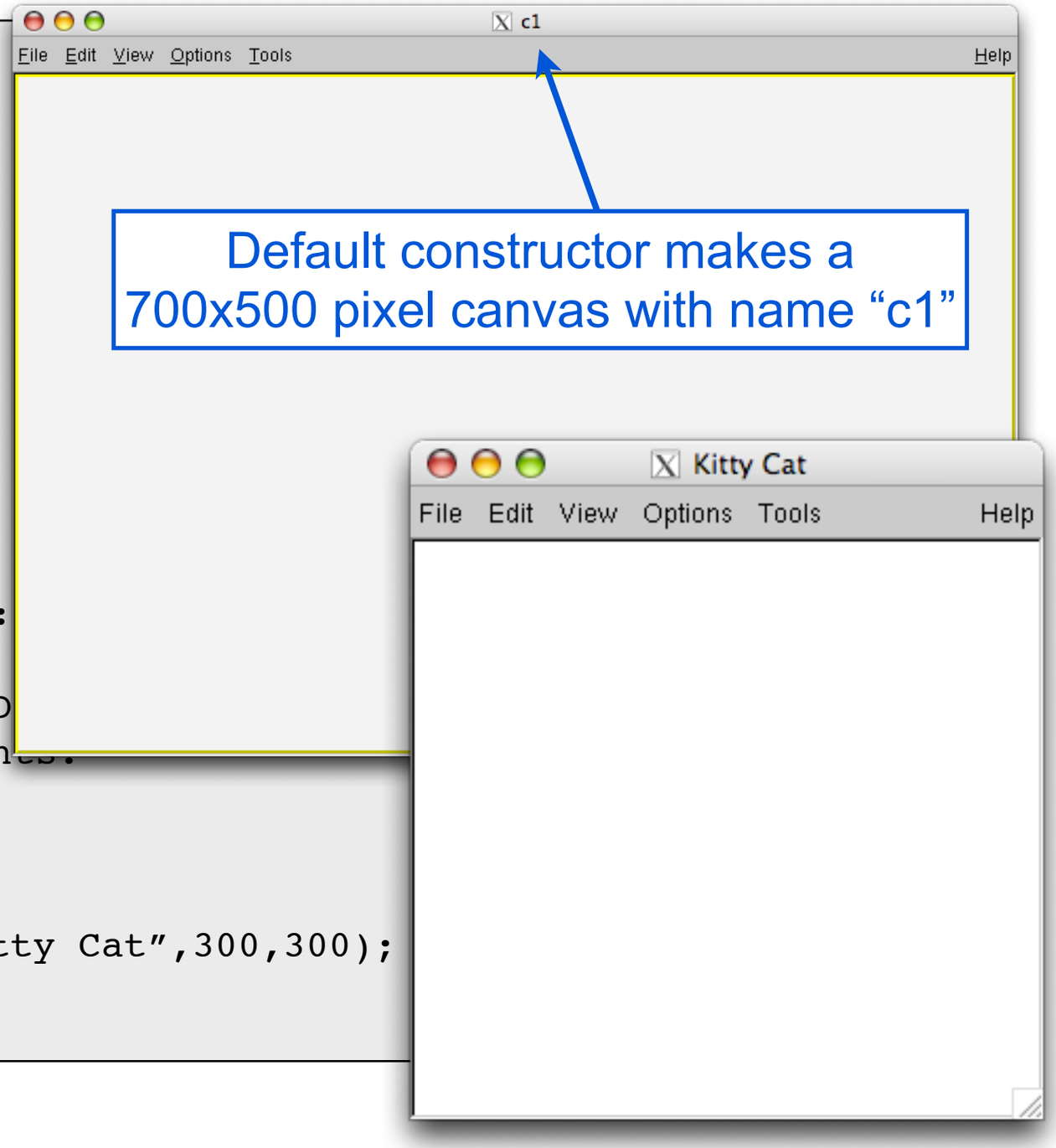
# Constructors, continued

```
javier:~> root
  ****************************************
  *                                      *
  *        W E L C O M E  to  R O O T    *
  *                                      *
  *   Version   5.24/00      29 June 2009  *
  *                                      *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch         *
  *                                      *
  ****************************************

ROOT 5.24/00 (trunk@29257, Jun 30 2009, 09:23:

CINT/ROOT C/C++ Interpreter version 5.17.00, D
Type ? for help. Commands must be C++ statemen
Enclose multiple statements between { }.

root [0] TCanvas *dog = new TCanvas();
root [1] TCanvas *cat = new TCanvas("cat","Kitty Cat",300,300);
```




Note: Commands you execute directly from the command line will be saved in a file called .root_hist (which will be in your home area for linux & mac... not sure where for windows).

# Unnamed Macros

Download and open the following file with your favorite editor:
http://home.fnal.gov/~jlraaf/2011REU/lecture1_exercises/unnamed_macro.C

Find out which directory you're in, then change to where you saved the macro.

```
root [0] gSystem->pwd();
root [1] gSystem->cd("/directory/where/you/put/downloaded/files");
root [2] .x unnamed_macro.C
Hello Duke REU students!
x = 31.2
y = 0
i = 12
The sum of x and i is: 43.2
root [3]
```

Execute the macro by "`.x`"

# Named Macros

Download and open the following file with your favorite editor:
http://home.fnal.gov/~jlraaf/2011REU/lecture1_exercises/named_macro.C

Load the macro by "`.L`"
Then execute the function
"`main()`" contained in it

```
root [3] .L named_macro.C
root [4] main()
Hello Duke REU students!
x = 31.2
y = 0
i = 12
The sum of x and i is: 43.2
root [5]
```

# Compiled Code

Download and open the following file with your favorite editor:
http://home.fnal.gov/~jlraaf/2011REU/lecture1_exercises/compiled_code.C

```
javier:~> g++ -Wall `root-config --cflags --libs` -o test compiled_code.C
javier:~> ls -l
-rwxr-xr-x  1 jlraaf  staff    13K Jul 17 14:39 test
-rw-r--r--  1 jlraaf  staff   777B Jul 17 14:24 compiled_code.C
-rw-r--r--  1 jlraaf  staff   674B Jul 17 13:23 named_macro.C
-rw-r--r--  1 jlraaf  staff   911B Jul 17 13:22 unnamed_macro.C
javier:~> ./test
Hello Duke REU students!
x = 31.2
y = 4.27382e-41
i = 12
The sum of x and i is: 43.2
javier:~>
```

Notice that ‘y’ is some crazy value.
It was not initialized by the compiler.
That’s okay for us here because we’ll assign it a value later that is the sum of x and i.
Just beware... this can sometimes cause nasty hard-to-find bugs.

Why should you use compiled code?

- For simple things it doesn’t matter, but for less simple things (your analysis, perhaps) it will run MUCH faster than using a named or unnamed macro in ROOT.
- It will force you to write proper C++

# Opening a file and accessing its contents

Download the following file:

http://home.fnal.gov/~jlraaf/2011REU/lecture1_exercises/histograms.root

```
root [0] TFile* f1 = new TFile("histograms.root");
root [1] f1->ls();
TFile**         histograms.root
 TFile*         histograms.root
  KEY: TH1F     histo1;1        Fancy 1-D Histogram
  KEY: TH2F     histo2;1        Schmancy 2-D Histogram
root [2] TCanvas *c1 = new TCanvas(<TAB>
TCanvas TCanvas(Bool_t build = kTRUE)
TCanvas TCanvas(const char* name, const char* title = "", Int_t form = 1)
TCanvas TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
TCanvas TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t ww,
Int_t wh)
TCanvas TCanvas(const char* name, Int_t ww, Int_t wh, Int_t winid)
root [3] TCanvas *c1 = new TCanvas(
```

List the contents of the file.

This file contains 2 histograms

Tip: If you can't remember the correct syntax, press "Tab" for ROOT to help you complete a command.

## Exercise 1:

Complete the command to make a canvas that is 600 x 300 pixels.

Divide it into 2 regions. (Hint: See TPad class definition at root.cern.ch and look for method Divide)

Change to region 1 and draw histo1. (Hint: c1->cd(1))

Change to region 2 and draw histo2.

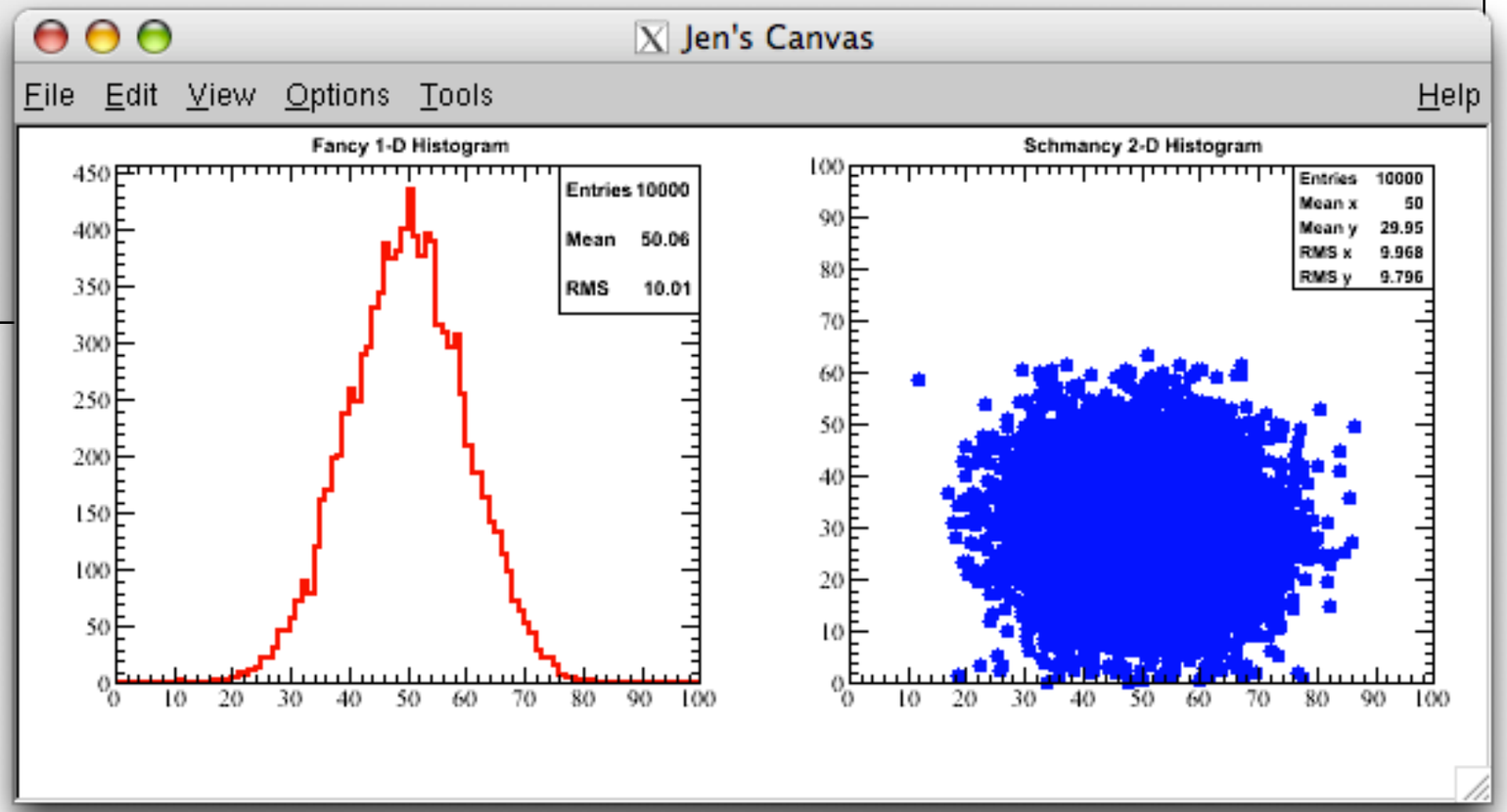
Note: If you don't make a canvas, ROOT will create one for you when you try to draw something. Personally, I like to make my own because I can control the dimensions.

# Exercise 1 Solution

Canvas name

Canvas title

```
root [3] TCanvas* c1 = new TCanvas("myc1","Jen's Canvas", 600,300);
root [4] myc1->Divide(1,2);
root [5] myc1->cd(1);
root [6] histo1->Draw();
root [7] myc1->cd(2);
root [8] histo2->Draw();
root [9]
```




ROOT manages memory using "names."
If you create two objects with the same name, even if they have different pointers, it will complain about memory leaks and delete one of your objects.

Exercise 2:

Try it to see what happens!

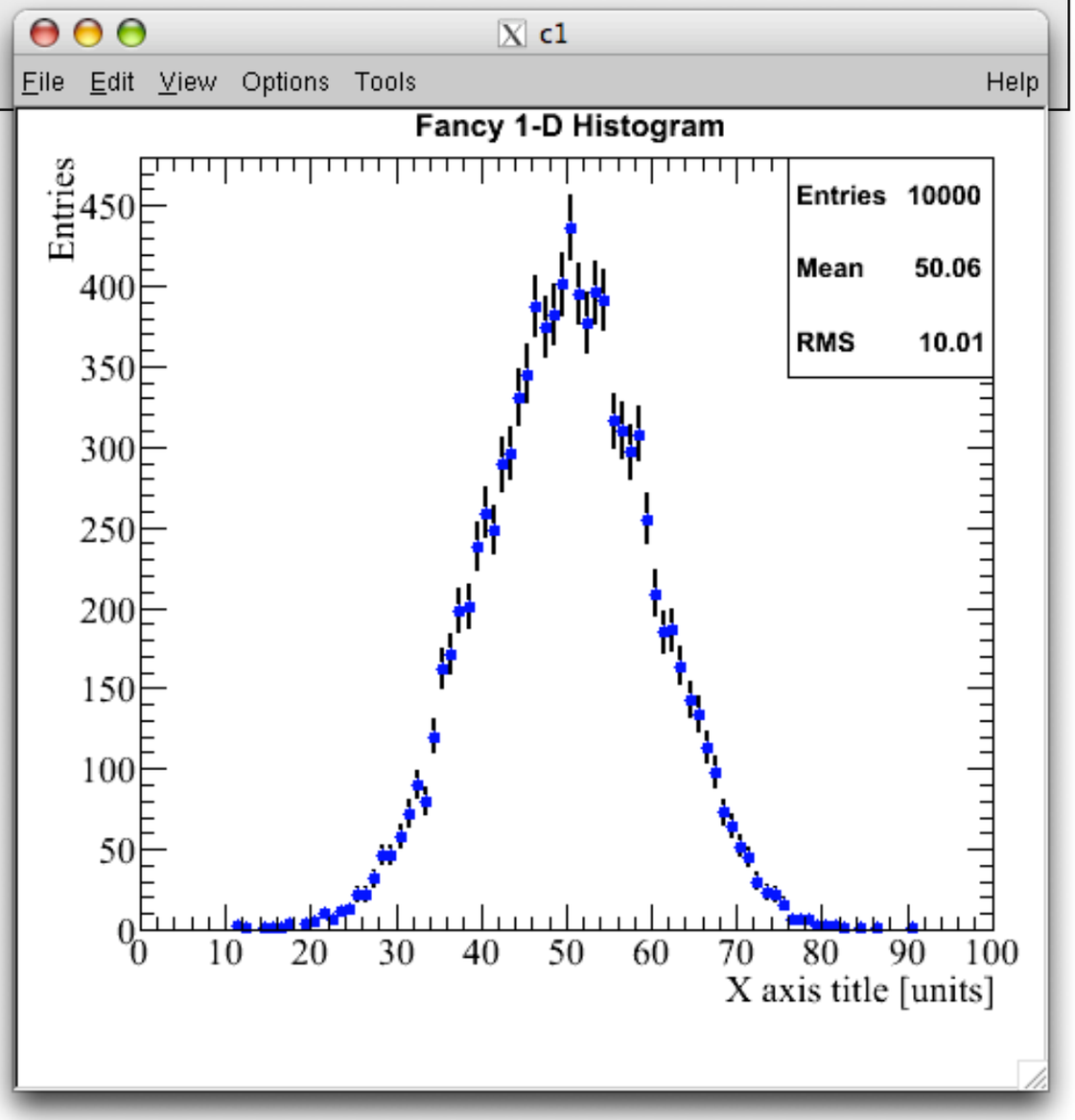
Make a new canvas with pointer c2, but the same name as your first canvas.

Try to change back to the first canvas.

# Changing Histogram Attributes

Histograms are drawn via the `THistPainter` class in ROOT.
You can find all drawing options by looking at the web documentation
http://root.cern.ch/root/html/THistPainter.html

```
root [9] histo1->SetLineColor(kBlack);
root[10] histo1->SetMarkerColor(kBlue);
root[11] histo1->GetXaxis()->SetTitle("X axis title [units]")
root[12] histo1->GetYaxis()->SetTitle("Entries");
root[13] histo1->Draw("ep");
```

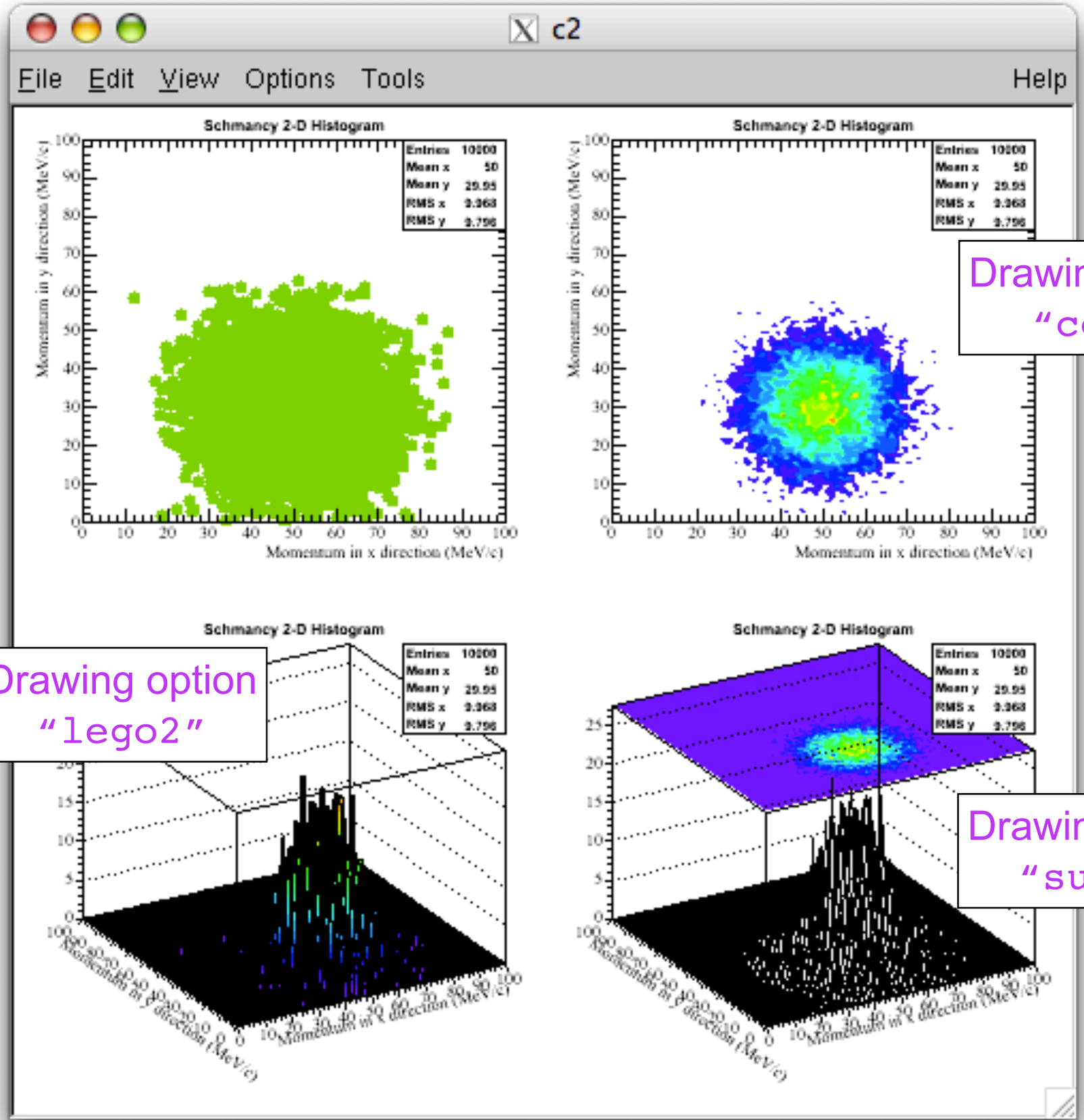



## Exercise 3:

- Change some attributes of histo2.
- Redraw it.
- Try other drawing options like "lego2"

Hint: Check THistPainter for even more drawing options on 2D histograms

# Exercise 3 (possible) Solution (of many)




You may have noticed that the default ROOT palette is hideous. Set this less offensive one by doing:

```
gStyle->SetPalette(1);
```

# Trees/Ntuples

Download the following file and open it in ROOT:

http://home.fnal.gov/~jlraaf/2011REU/lecture1_exercises/tree.root

```
root [0] TFile *myfile = new TFile("tree.root");
root [1] myfile->ls();
TFile**         tree.root
 TFile*         tree.root
  KEY: TTree    tree1;1 Reconstruction ntuple
root [2] TTree *t1 = (TTree *)myfile.Get("tree1");
root [3] t1->Print();
******************************************************************************
*Tree    :tree1     : Reconstruction ntuple                                  *
*Entries :   100000 : Total =         2810647 bytes  File  Size =    2171135 *
*       :          : Tree compression factor =   1.30                       *
******************************************************************************
*Br    0 :event     : event/I                                                *
*Entries :   100000 : Total  Size=     421248 bytes  File Size  =     134514 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   2.85     *
*.............................................................................*
*Br    1 :ebeam     : ebeam/F                                                *
*Entries :   100000 : Total  Size=     421248 bytes  File Size  =     260330 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.47     *
*.............................................................................*
*Br    2 :px        : px/F                                                   *
*Entries :   100000 : Total  Size=     421194 bytes  File Size  =     359238 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.07     *
*.............................................................................*
*Br    3 :py        : py/F                                                   *
*Entries :   100000 : Total  Size=     421194 bytes  File Size  =     359138 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.07     *
*.............................................................................*
*Br    4 :pz        : pz/F                                                   *
*Entries :   100000 : Total  Size=     421194 bytes  File Size  =     292046 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.31     *
*.............................................................................*
*Br    5 :zv        : zv/F                                                   *
*Entries :   100000 : Total  Size=     421194 bytes  File Size  =     349087 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.10     *
*.............................................................................*
*Br    6 :chi2      : chi2/F                                                 *
*Entries :   100000 : Total  Size=     421230 bytes  File Size  =     321049 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.20     *
*.............................................................................*
```

ROOT data usually stored in a TTree (or simplified version: TNtuple)

More versatile than histograms (no information loss).

For a simple tree/ntuple structure, you can think of it as a table.

If each "TBranch" is like a column, then each "Entry" is a new cell in the column.

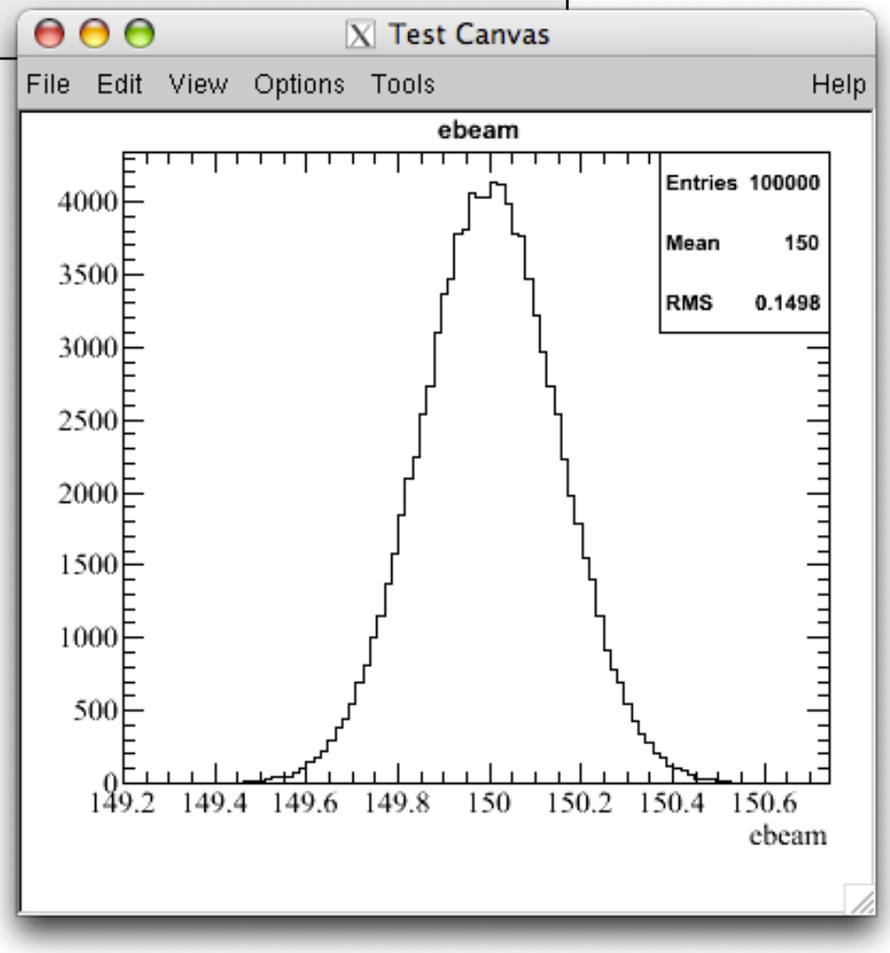
Print structure of tree to screen.

This tree contains 7 variables:

`event, ebeam, px, py, pz, zy, chi2`

# Trees/Ntuples

```
root [4] TCanvas *jen = new TCanvas("jen","Test Canvas",400,400);
root [5] t1->Draw("ebeam");
```




## Exercise 4:

Draw some other variables from the tree.

Draw a 2D plot of px vs. py

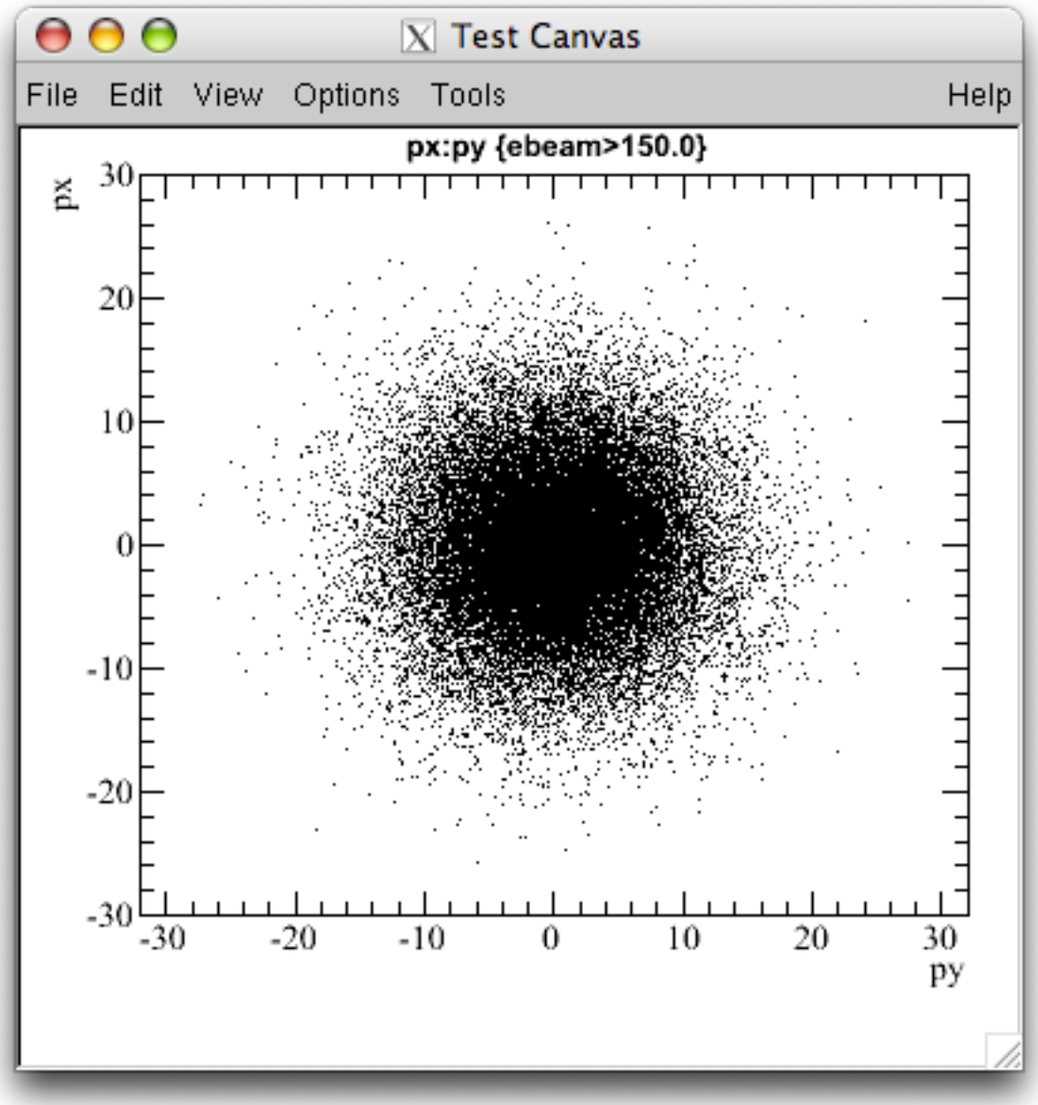
Hint: Look at the TTree Draw methods at http://root.cern.ch/root/html530/TTree.html

Draw px vs. py for events with ebeam > 150.0

Hint: The TTree Draw methods can take a "selection" character string

# Exercise 4 Solution

```
root [6] t1->Draw("px:py","ebeam>150.0");
```



# Projecting from a tree into a histogram

Sometimes you may want to put a variable from a tree into a histogram.

First define the histogram:

Name, Title, Number of bins, Low edge, High edge

```
root [7] TH1F *h_ebeam = new TH1F("h_ebeam","Beam energy",100,149.0,151.0);
```

Then use the `TTree` method "`Project`" to put the tree contents into the histogram:

```
root [8] t1->Project("h_ebeam", "ebeam", "(px > 10.0) || (py <= 5.0)");
```

Selection cuts: optional argument

To define complicated or often-used cuts:

```
TCut* cut1 = new TCut("px > 10.0");
TCut* cut2 = new TCut("py <= sqrt(2+px**2)");
TCut* cut3 = new TCut(*cut1 && *cut2);
```

Then use the TCut when you draw!

```
t1->Draw("ebeam", *cut3);
```

Cuts are specified using C logic

| | |
|---|---|
| && | AND |
| \|\| | OR |
| == | equal |
| != | NOT equal |
| > | greater than |
| < | less than |
| >= | greater or equal to |
| <= | less or equal to |

Exercise 5:
Try making some cuts on tree variables and drawing/projecting.

# Reading data from a text file to make a tree

Download the following file:

http://home.fnal.gov/~jlraaf/2011REU/lecture1_exercises/input.txt

readdata.C

File Edit Options Buffers Tools C++ Help

```
{

  // Make a new tree and fill it with information from file
  TTree *mytree = new TTree("jen","Crude Oil Prices");
  mytree->ReadFile("input.txt","year/I:cost/F");


  // Make a new output file (recreate it if it already exists)
  TFile *fout = new TFile("oilprices.root","Recreate");
  fout->cd();

  // Write the tree into the file
  mytree->Write();

  // Close the file
  fout->Close();

  // Clean up memory usage.
  delete mytree;
  delete fout;

}
```

-U:---  readdata.C  All (17,0)  (c++/l Abbrev)--11:49PM 0.41----------

Quit

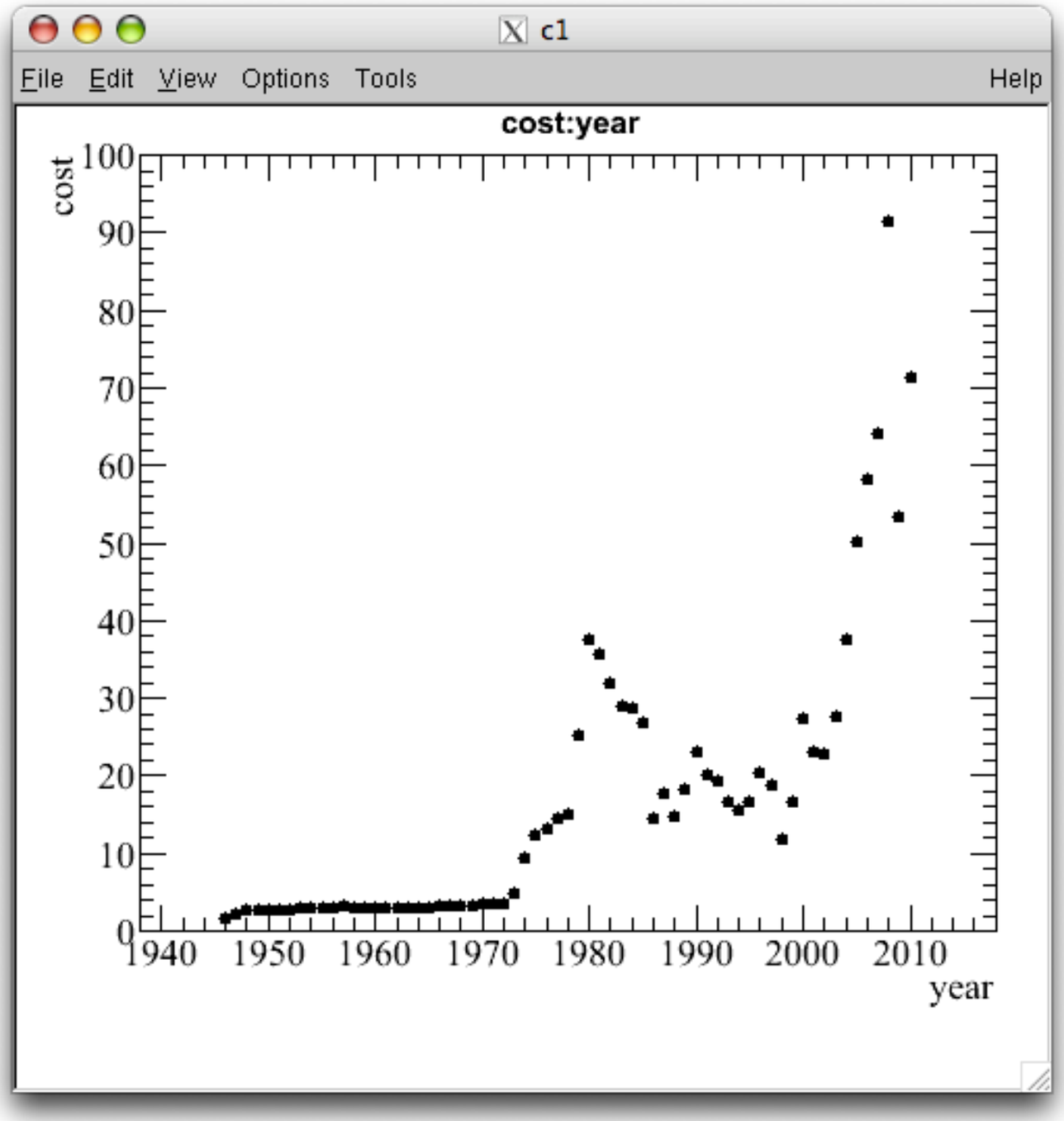
Exercise 6:

Create a new macro and edit it to do these commands.

Execute it from the ROOT command line.

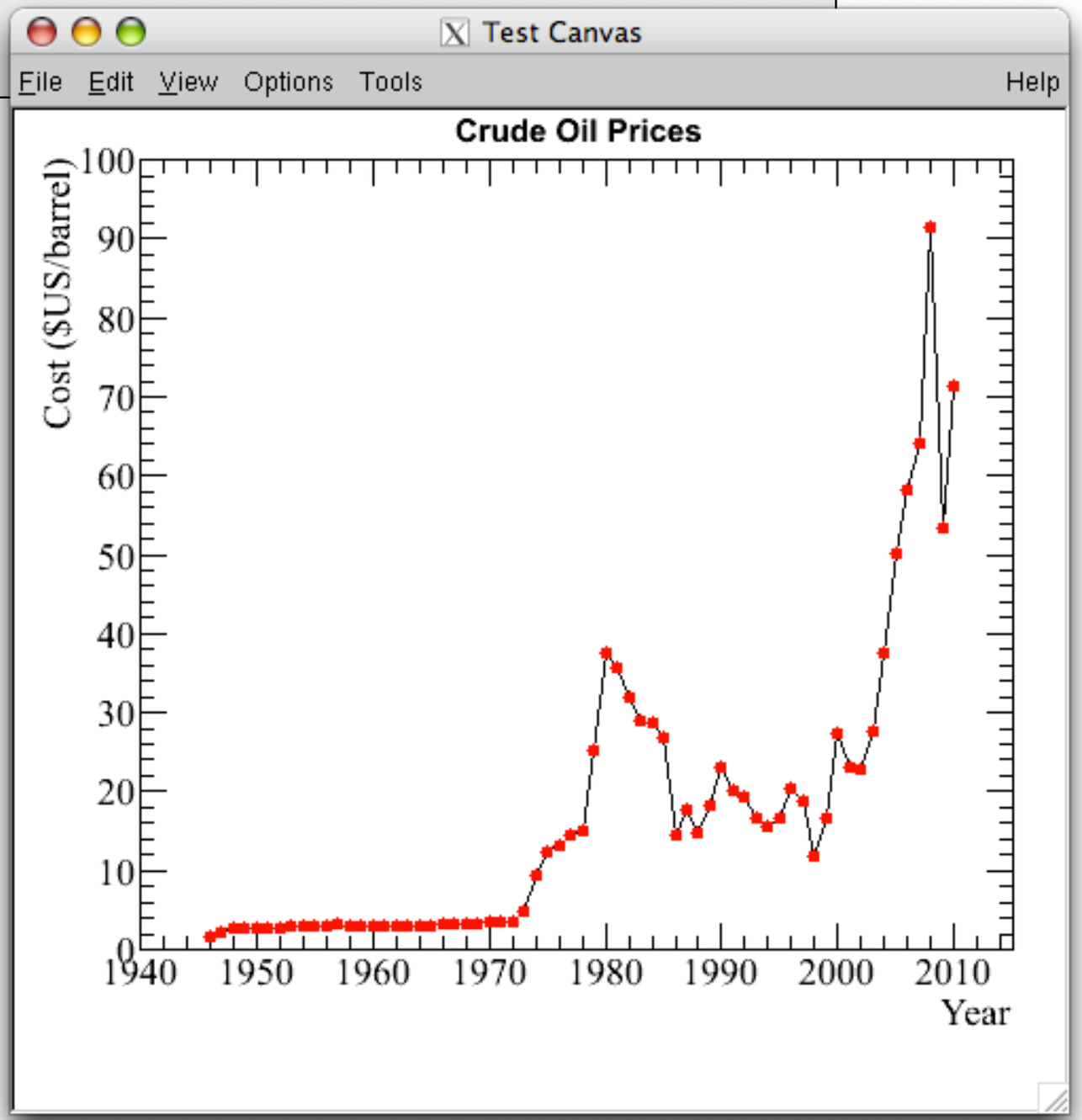
Open the newly created ROOT file (oilprices.root) and plot cost vs. year

# Exercise 6 Solution



# Another way to read in your data

```
root [0] TCanvas *jen = new TCanvas("jen","Test Canvas",500,500);
root [1] TH1F *frame = (TH1F *)jen->DrawFrame(1930,0.0,2015,100.0);
root [2] frame->SetTitle("Crude Oil Prices");
root [3] frame->GetXaxis()->SetTitle("Year");
root [4] frame->GetYaxis()->SetTitle("Cost ($US/barrel)");
root [5] TGraph *mygraph = new TGraph("input.txt");
root [6] mygraph->Draw();
root [7] mygraph->Draw("p");
```



# Homework

Learn how to define a 1-dimensional function (TF1).
Make one that is sin(x)/x and draw it.

# Next Time...

## TF1 Fitting

# Extra

# Warning...

Interactive ROOT uses a C++ interpreter (CINT) which allows (but does not require) you to write *pseudo*-C++

Be careful! This will make your programming much more difficult later in life!

It's best if you try to use standard C++ syntax, instead of the CINT shortcuts.

ROOT CINT syntax allows the following sloppy things:

"**.**" and "**->**" are interchangeable

"**;**" is optional at the end of single commands

Many commands may be accessed interactively (point and right-click in plots)

Don't be sloppy!

# Quitting!

Possibly the most asked question from new ROOT users is “How do I quit?!?!?”
The answer (which you’d never guess in a million years): “`.q`”

And if it’s stuck doing something and won’t pay attention to you, start adding extra “q”s!

- `.q` (quit)
- `.qqq` (Quit, I mean it!)

...

- `.qqqqqq` (HEY! I SAID QUIT!)
- `.qqqqqqq` (QUIT RIGHT NOW, OR ELSE!)