Introduction to Unix, C++ ,and Root

INTRODUCTION

- Today we will start to introduce the technical bits you will need to learn

- My approach will be pragmatic - there are 1000 page books on C++ that you can use as references but you can literally make a career out of it

- Here we will introduce some basics and then add on more things as we go along during the series of lectures

UNIX - WHAT IS IT

- UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

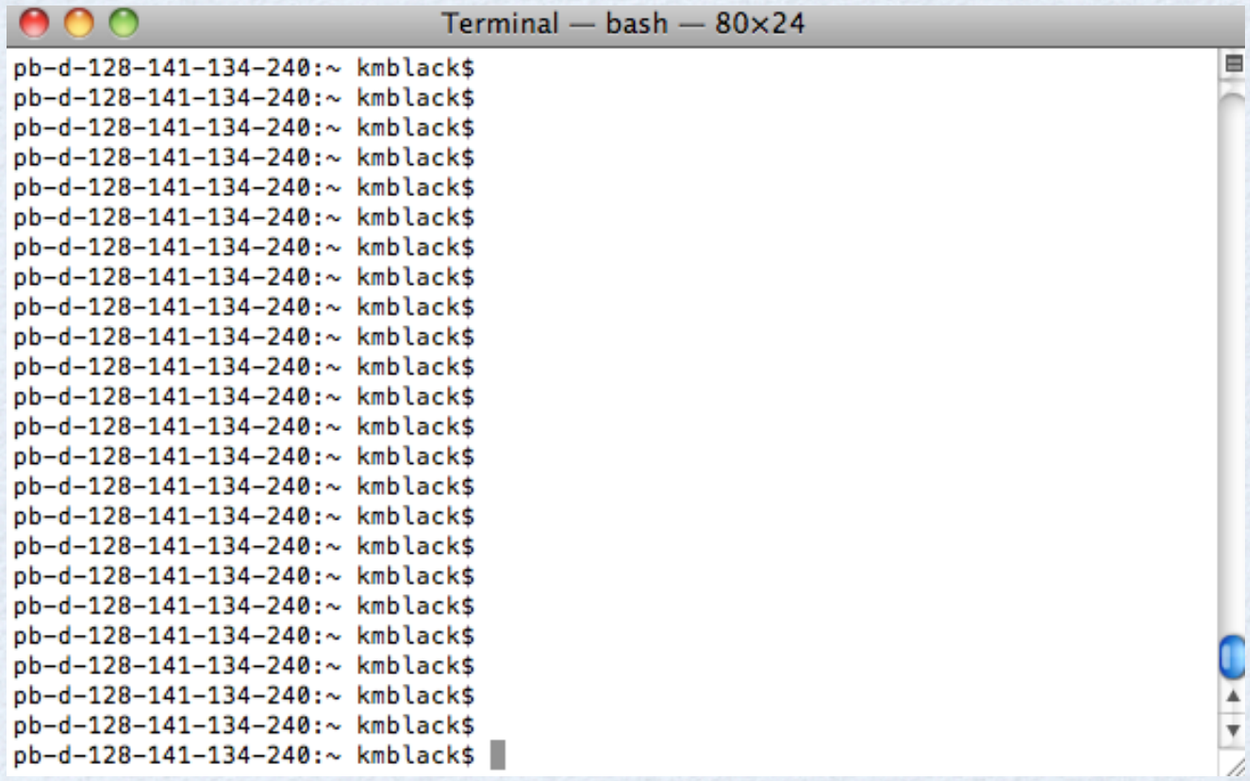
- Modern implementations can have graphical interfaces - we won't be talking about these much as they are mostly self explanatory

- We will review what UNIX is and how to do some basics. I will then show you some references for more information - as with C++ there is a rabbit hole to go down if you choose but we will focus on functional use of the tools

What is Unix

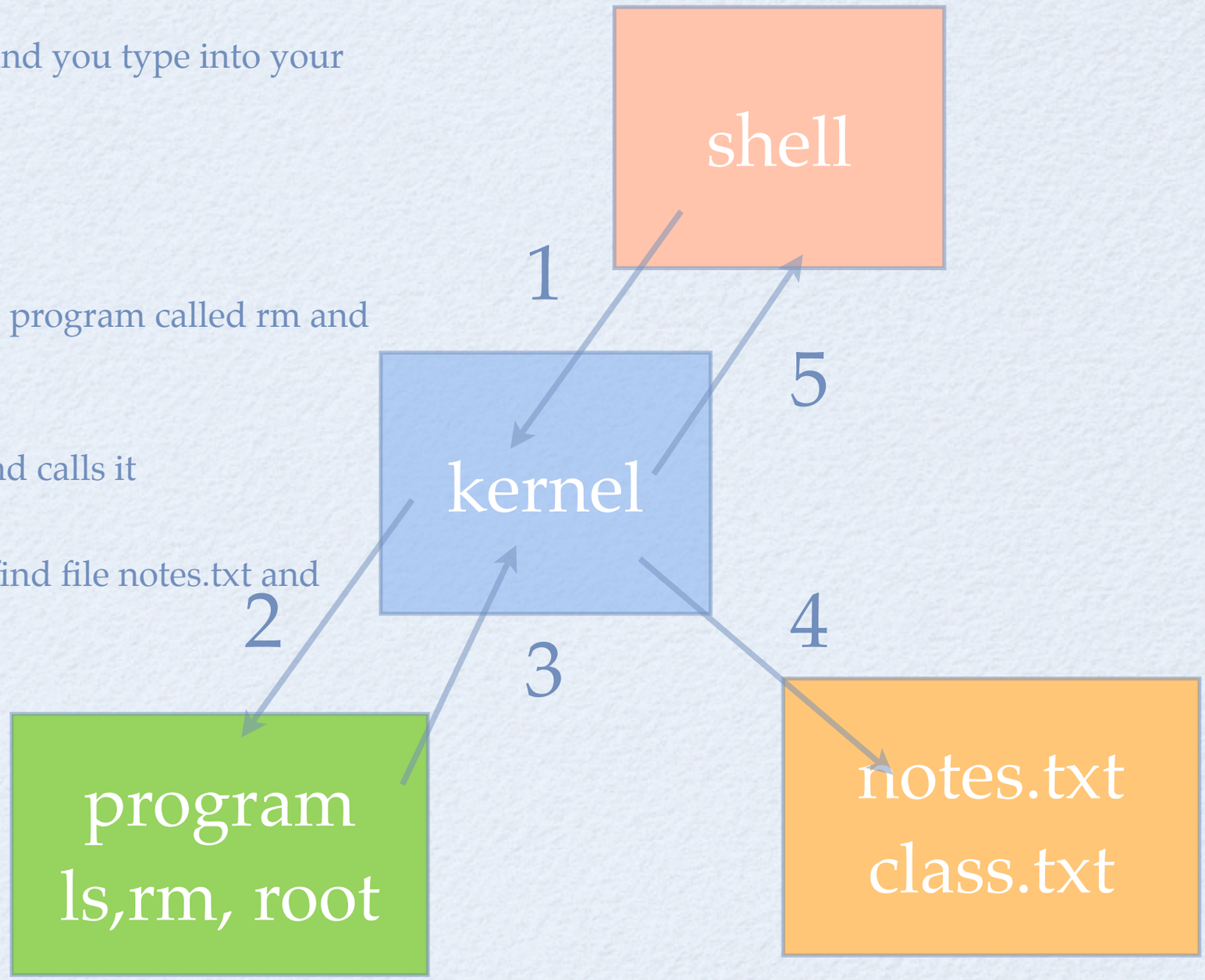
Composed of three elements

- Kernel
  - `hub of the operating system, allocates time for the cpu , memory, handles files, does underlying work for all operations
- The Shell
  - an interface between the user and the kernel. This is the program that is running when you log into a terminal
- Programs
  - the instructions that you ask the computer to follow



HOW DOES THIS WORK

- if you have a file called notes.txt and you type into your terminal
  - rm notes.txt
- (1) shell requests kernel to go find program called rm and pass it the argument `notes.txt'
- (2) kernel searches for program and calls it
- (3) program rm tells kernel to go find file notes.txt and delete it
- (4) kernel does it
- (5) kernel returns control to shell

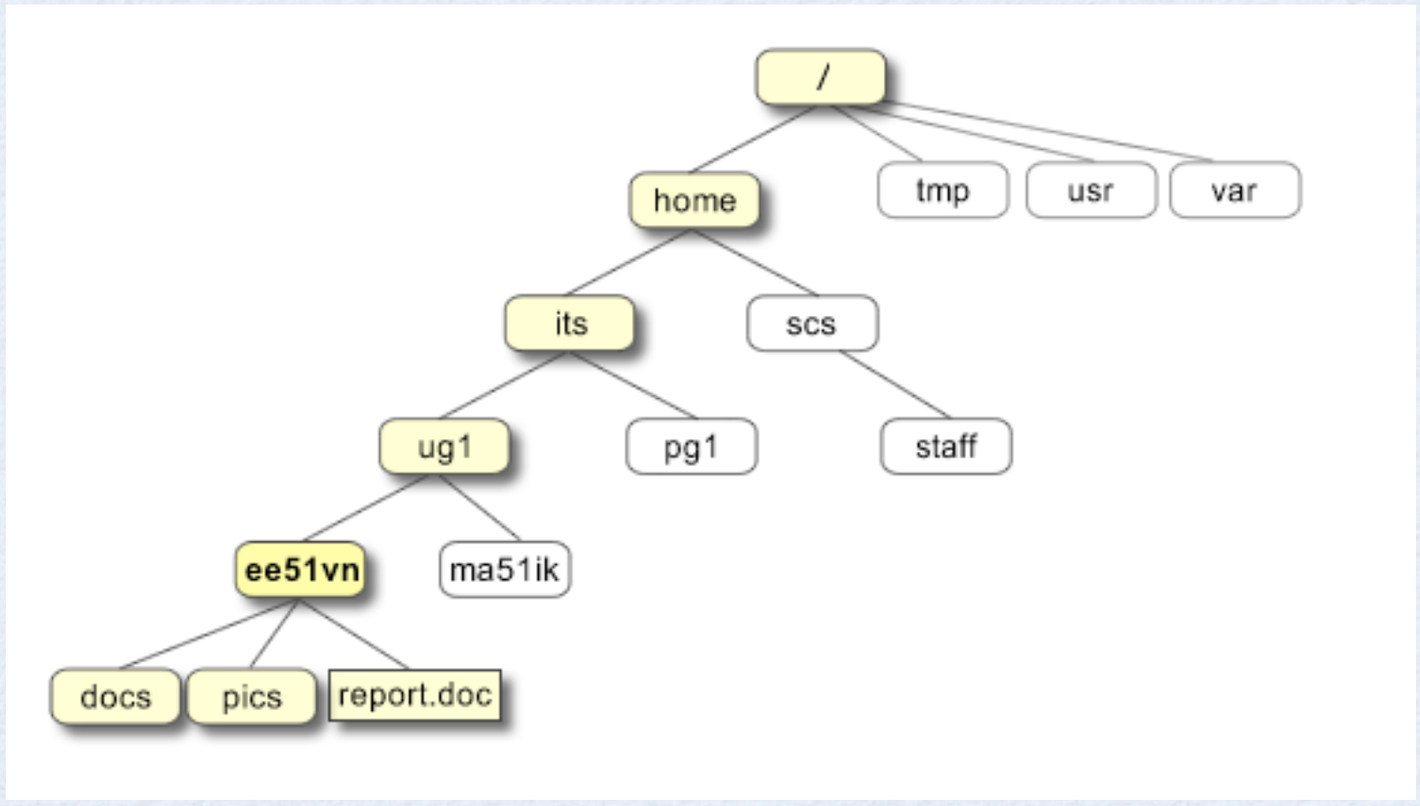


SHELLS

- There are several shells Bourne, C, Z shell
  - roughly they are equivalent, small differences in the syntax of some commands
  - on most machines there are a few different shells that
- To find out what shell you are using type
  - echo $SHELL

FILES AND PROCESSES

- Everything in unix is either a file or a process
  - a process is just an executing program (in unix each gets a PID - process ID)
  - a file is a collection of information
    - a text file, source code for a program, a directory



BASIC UNIX COMMANDS

- Open a unix terminal (if you run linux you know how to do this, on a mac you run the program terminal)
- For windows machines you will need an ssh client
  - http://www.openssh.org/windows.html
  - http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Do this now or work with a friend today if you don't have one
- We will go through a series of basic commands that will allow you to get started . if you already have experience with unix based systems this will be review

FILES AND DIRECTORIES

- Listing files
  - When you first login to a unix machine you will be in your home directory
  - To find out what is in your home directory type
    - ls
    - ls -a (`hidden files')
  - make a directory
    - mkdir UnixStuff
  - change directory
    - cd UnixStuff
  - print the current directory
    - pwd

```
pb-d-128-141-134-240:~ kmblack$ ls -a
.
..
.CFUserTextEncoding
.DS_Store
.KoalaNext
.SMART-customer-experience-program-shared-uuid
.Trash
.Xauthority
.Xcode
.Xdefaults
.Xresources
.anyconnect
.bash_history
.config
.cups
```

```
pb-d-128-141-134-240:~ kmblack$ mkdir UnixStuff
pb-d-128-141-134-240:~ kmblack$ cd UnixStuff/
pb-d-128-141-134-240:UnixStuff kmblack$ ls
pb-d-128-141-134-240:UnixStuff kmblack$
```

```
pb-d-128-141-134-240:UnixStuff kmblack$ pwd
/Users/kmblack/UnixStuff
```

SIMPLE FILE COMMANDS

Command	Meaning
ls	list files and directories
ls -a	list all files and directories
mkdir	make a directory
cd *directory*	change to named directory
cd	change to home-directory
cd ~	change to home-directory
cd ..	change to parent directory
pwd	display the path of the current directory

to first order these are the commands you need to know to look at and go through a unix directory and file structure

MORE WITH FILES

Command	Meaning
cp *file1 file2*	copy file1 and call it file2
mv *file1 file2*	move or rename file1 to file2
rm *file*	remove a file
rmdir *directory*	remove a directory
cat *file*	display a file
less *file*	display a file a page at a time
head *file*	display the first few lines of a file
tail *file*	display the last few lines of a file
grep *'keyword' file*	search a file for keywords
wc *file*	count number of lines/words/characters in file

you don't have
to memorize these
but you will remember
these basic ones
because you use
them so
often

DIRECTING OUTPUT

- One of the great advantages of unix is the ability to redirect output from one command into the input of another

- I will show a few basic example of why this is useful

- it turns out that you can combine many of these commands to do a lot with a single command!

- Most output
- of unix commands goes to the `standard output'
  - ie. your terminal screen
- however you can redirect this to files or other commands
- for example consider the cat command

```
pb-d-128-141-134-240:UnixStuff kmblack$ cat > list1
pear
banana
apple
plum
pb-d-128-141-134-240:UnixStuff kmblack$ less list1

pear
banana
apple
plum
list1 (END)
```

- do the following type
  - cat > list1
- then type in the names of some fruit pressing return after each one
- at the end push control D

APPENDING A FILE

- You can append a file by doing
  - cat >> list1
- and then typing more fruit
- can merge two files by directing the output of cat on two files into a third!

```
pb-d-128-141-134-240:UnixStuff kmblack$ cat list1
pear
banana
apple
plum
peach
grape
orange
pb-d-128-141-134-240:UnixStuff kmblack$
```

```
pb-d-128-141-134-240:UnixStuff kmblack$ ls
list1   list2
pb-d-128-141-134-240:UnixStuff kmblack$ cat list1 list2 > biglist
pb-d-128-141-134-240:UnixStuff kmblack$ cat biglist
pear
banana
apple
plum
peach
grape
orange
apricot
kiwi
pb-d-128-141-134-240:UnixStuff kmblack$
```

INPUT

- use the < symbol to redirect the input
- for example the sort command knows how to alphabetize
- if you type
  - sort
- and then
  - dog
  - cat
  - bird
  - ape
  - control D

```
pb-d-128-141-134-240:UnixStuff kmblack$ sort
dog
cat
bird
ape^D
ape
bird
cat
dog
```

PIPES

- If you are on a multiuser system you can see who else is on the system
  - who

```
[kblack@lxplus255]~%
[kblack@lxplus255]~% who
wittgen  pts/0        2012-01-16 22:51 (173-228-112-199.dsl.dynamic.sonic.net)
maliev   pts/2        2012-01-25 21:16 (f052142026.adsl.alicedsl.de)
kblack   pts/3        2012-01-26 00:10 (aannecy-651-1-76-64.w86-209.abo.wanadoo.fr)
maliev   pts/5        2012-01-25 15:48 (natr.physik.hu-berlin.de)
dgrandi  pts/6        2012-01-24 17:00 (amsmc04.mib.infn.it)
itopsisg pts/7        2012-01-25 17:26 (dhcp167.physics.ntua.gr)
peiffer  pts/8        2012-01-25 18:21 (ekplx59.physik.uni-karlsruhe.de)
itopsisg pts/11       2012-01-24 19:59 (143.233.252.2)
maliev   pts/12       2012-01-25 21:17 (f052142026.adsl.alicedsl.de)
mwhitehe pts/13       2012-01-25 14:50 (iseran.epp.warwick.ac.uk)
shsun    pts/14       2012-01-16 17:22 (pb-d-128-141-35-189.cern.ch)
paul     pts/16       2012-01-25 22:58 (lns-bzn-50f-81-56-198-137.adsl.proxad.net)
peiffer  pts/19       2012-01-25 13:00 (ekplx59.physik.uni-karlsruhe.de)
louis    pts/10       2012-01-23 07:10 (jul74-1-88-186-228-163.fbx.proxad.net)
knikolic pts/23       2012-01-25 23:55 (195.43.57.14)
khotilov pts/24       2012-01-23 19:34 (hepcms1.physics.tamu.edu)
masmith  pts/25       2012-01-23 17:11 (heppc2-sl53.hep.manchester.ac.uk)
suyogs   pts/26       2012-01-25 23:55 (aannecy-158-1-65-170.w90-52.abo.wanadoo.fr)
fballi   pts/27       2012-01-25 23:55 (160.235.116.78.rev.sfr.net)
kdziedzi pts/29       2012-01-16 10:13 (pb-d-128-141-28-14.cern.ch)
florian  pts/32       2012-01-25 23:55 (pb-d-128-141-235-128.cern.ch)
yangyong pts/35       2012-01-18 22:28 (pccityongnew.cern.ch)
maliev   pts/36       2012-01-24 17:34 (natr.physik.hu-berlin.de)
sonnen   pts/37       2012-01-25 09:13 (pcac3-5.cern.ch)
kaiwu    pts/38       2012-01-25 14:54 (pcamsr0.cern.ch)
shimpei  pts/1        2012-01-21 19:20 (33-56.5-85.cust.bluewin.ch)
mwhitehe pts/40       2012-01-25 14:20 (iseran.epp.warwick.ac.uk)
flowerde pts/43       2012-01-23 11:37 (pcatlas57.mppmu.mpg.de)
kaiwu    pts/44       2012-01-25 14:55 (pcamsr0.cern.ch)
asakharo pts/56       2012-01-24 12:14 (pcth199.cern.ch)
ams      pts/60       2012-01-19 15:33 (pcamspg00.cern.ch)
ptedesco pts/30       2012-01-24 11:52 (localhost:27.0)
azzi     pts/68       2012-01-24 15:20 (pcpd01.cern.ch)
```

PIPES

- who > names.txt
- sort < names.txt
- which does the same thing as
  - who | sort
- the pipe command “pipes” or sends the output of one command into the other
  - In this case first executes who
  - then pipes the output of who into sort
  - and then sort returns the output to the standard output

```
[kblack@lxplus255]~% sort < names.txt
ams      pts/60       2012-01-19 15:33 (pcamspg00.cern.ch)
asakharo pts/56       2012-01-24 12:14 (pcth199.cern.ch)
azzi     pts/68       2012-01-24 15:20 (pcpd01.cern.ch)
dgrandi  pts/6        2012-01-24 17:00 (amsmc04.mib.infn.it)
fballi   pts/27       2012-01-25 23:55 (160.235.116.78.rev.sfr.net)
flowerde pts/43       2012-01-23 11:37 (pcatlas57.mppmu.mpg.de)
itopsisg pts/11       2012-01-24 19:59 (143.233.252.2)
itopsisg pts/7        2012-01-25 17:26 (dhcp167.physics.ntua.gr)
kaiwu    pts/38       2012-01-25 14:54 (pcamsr0.cern.ch)
kaiwu    pts/44       2012-01-25 14:55 (pcamsr0.cern.ch)
kblack   pts/3        2012-01-26 00:10 (aannecy-651-1-76-64.w86-209.abo.wanadoo.fr)
kdziedzi pts/29       2012-01-16 10:13 (pb-d-128-141-28-14.cern.ch)
khotilov pts/24       2012-01-23 19:34 (hepcms1.physics.tamu.edu)
knikolic pts/23       2012-01-25 23:55 (195.43.57.14)
louis    pts/10       2012-01-23 07:10 (jul74-1-88-186-228-163.fbx.proxad.net)
maliev   pts/12       2012-01-25 21:17 (f052142026.adsl.alicedsl.de)
maliev   pts/2        2012-01-25 21:16 (f052142026.adsl.alicedsl.de)
maliev   pts/36       2012-01-24 17:34 (natr.physik.hu-berlin.de)
maliev   pts/5        2012-01-25 15:48 (natr.physik.hu-berlin.de)
masmith  pts/25       2012-01-23 17:11 (heppc2-sl53.hep.manchester.ac.uk)
mwhitehe pts/13       2012-01-25 14:50 (iseran.epp.warwick.ac.uk)
mwhitehe pts/40       2012-01-25 14:20 (iseran.epp.warwick.ac.uk)
peiffer  pts/19       2012-01-25 13:00 (ekplx59.physik.uni-karlsruhe.de)
peiffer  pts/8        2012-01-25 18:21 (ekplx59.physik.uni-karlsruhe.de)
ptedesco pts/30       2012-01-24 11:52 (localhost:27.0)
ptedesco pts/42       2012-01-24 13:19 (localhost:27.0)
shimpei  pts/1        2012-01-21 19:20 (33-56.5-85.cust.bluewin.ch)
shsun    pts/14       2012-01-16 17:22 (pb-d-128-141-35-189.cern.ch)
sonnen   pts/37       2012-01-25 09:13 (pcac3-5.cern.ch)
suyogs   pts/26       2012-01-25 23:55 (aannecy-158-1-65-170.w90-52.abo.wanadoo.fr)
wittgen  pts/0        2012-01-16 22:51 (173-228-112-199.dsl.dynamic.sonic.net)
yangyong pts/35       2012-01-18 22:28 (pccityongnew.cern.ch)
```

SUMMARY OF COMMANDS

- Note that you can do multiple piles for various unix commands
- This allows one to do very complicated things in a very short (but not simple) set of commands

Command	Meaning
command > file	redirect standard output to a file
command >> file	append standard output to a file
command < file	redirect standard input from a file
command1 | command2	pipe the output of command1 to the input of command2
cat *file1 file2 > file0*	concatenate file1 and file2 to file0
sort	sort data
who	list users currently logged in

MORE INFORMATION

- That is enough information to get you navigating through and familiar with basic unix files, directories, and processes

- **Unix: The Complete Reference**

- just search in google for unix references - there are many!

C++

- For many years physicists used FORTRAN as the basic computing language
- In the mid 90s HEP migrated to a new programming language we will touch on that today
- C++ is:
  - Object Oriented - We will discuss what this means exactly
  - Compiled - this means the code is optimized by a program called (not surprisingly) a compiler and checked for mistakes before hand
  - Complicated - it is a very powerful language - but also one that allows you to hang yourself with the rope it gives you if you aren't careful

What does a C++ program look like

// starting a line tells the compiler to ignore this line

get code from another file

All true C++ programs have a main function which is the start of the program and the end. Everything happens in between

the main() program returns an integer in in order to return control we have to **return** some int

```
//include headers; these are modules that include functions that you may use in your
 //program; we will almost always need to include the header that
// defines cin and cout; the header is called iostream.h
#include <iostream.h>

int main() {

//variable declaration
//read values input from user
//computation  and print output to user
return 0;
}
```

After you write a C++ program you compile it; that is, you run a program called **compiler** that checks whether the program follows the C++ syntax

- if it finds errors, it lists them
- If there are no errors, it translates the C++ program into a program in machine language which you can execute

NOTES

- what follows after // on the same line is considered comment
- indentation is for the convenience of the reader; compiler ignores all spaces and new line ; the delimiter for the compiler is the semicolon
- all statements ended by semicolon
- Lower vs. upper case matters!!
- Void is different than void
- Main is different that main

VARIABLE DECLATION

type variable-name;

Meaning: variable <variable-name> will be a variable of type <type>

Where type can be:

- int //integer
- double //real number
- char //character

Example:

```
int a, b, c;
double x;
int sum;
char my-character;
```

VARIABLE INPUT

cin >> variable-name;

Meaning: read the value of the variable called <variable-name> from the user

Example:

```
cin >> a;
cin >> b >> c;
cin >> x;
cin >> my-character;
```

VARIABLE OUTPUT

cout << variable-name;

Meaning: print the value of variable <variable-name> to the user

cout << "any message ";

Meaning: print the message within quotes to the user

cout << endl;

Meaning: print a new line

Example:

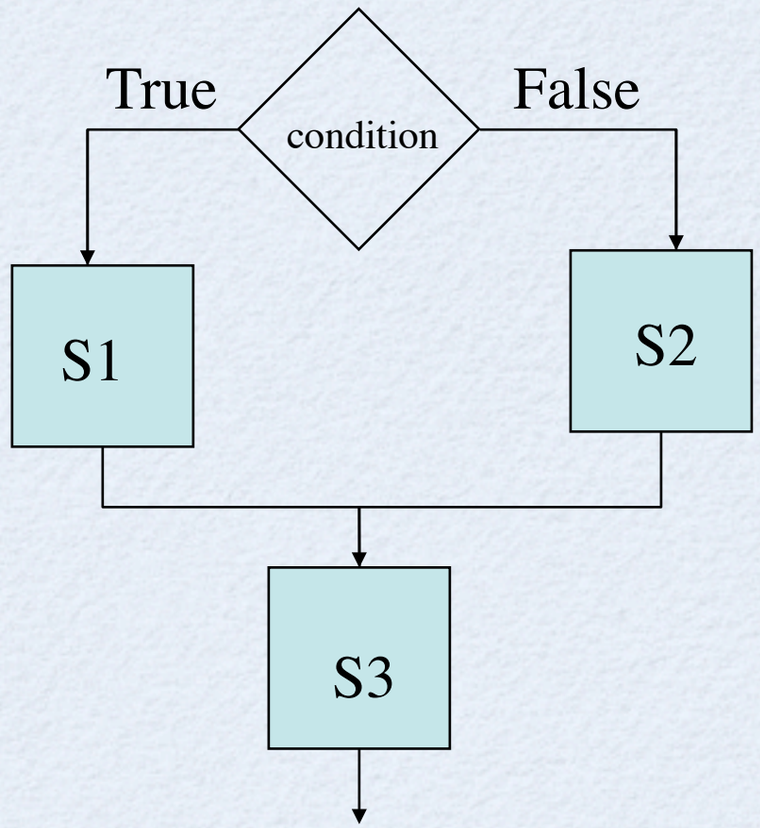
```
cout << a;
cout << b << c;
cout << "This is my character: " << my-character << " he he he"
        << endl;
```

PROGRAM CONTROL

- often we would like to control the flow of a program depending on a condition that we occur
- there are several C++ constructions that allow us to do just that

IF STATEMENTS

```
if (condition) {
    S1;
 }
 else {
    S2;
 }
 S3;
```



COMPARISONS

..are built using

- Comparison operators

  == equal

  != not equal

  < less than

  > greater than

  <= less than or equal

  >= greater than or equal

- Boolean operators

  && and

  || or

  ! not

EXAMPLES

Assume we declared the following variables:

```
int a = 2, b=5, c=10;
```

Here are some examples of boolean conditions we can use:

- `if (a == b) ...`
- `if (a != b) ...`
- `if (a <= b+c) ...`
- `if(a <= b) && (b <= c) ...`
- `if !((a < b) && (b<c)) ...`

IF EXAMPLE

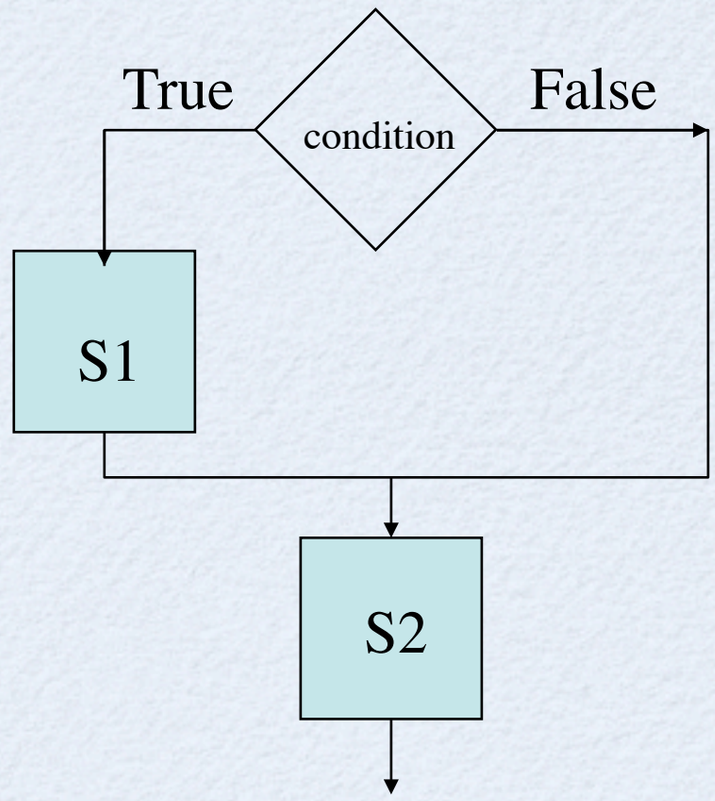
```
#include <iostream.h>

void main() {
int a,b,c;
cin >> a >> b >> c;

if (a <=b) {
        cout << “min is “ << a << endl;
        }
else {
        cout << “ min is “ << b << endl;
}
cout << “happy now?” << endl;
}
```

WHILE STATEMENTS

```
while (condition) {
   S1;
}
S2;
```



WHILE EXAMPLE

```
//read 100 numbers from the user and output their sum
#include <iostream.h>

void main() {
int i, sum, x;
sum=0;
i=1;
while (i <= 100) {
    cin >> x;
    sum = sum + x;
    i = i+1;
}
cout << “sum is “ << sum << endl;
}
```

FOR LOOP

- for (inital statement, expression, increment statement) {statement;}

```
#include<iostream.h>

const int MAXNUMBERS=6;
int main( )
{
 int i, number[MAXNUMBERS];
 for(i=0;i<=MAXNUMBERS;  i++) //Enter the numbers
    {
    cout<<"Enter  a number :"
    cin>>number[i];
    }
cout<<endl<<endl;
for(i=0; i<MAXNUMBERS;i++)// Print the numbers
    {
    cout<<"Number "<<number[i]<<"is" <<NUMBERS[i]<<endl;
    return 0;
    }
}
```

ARITHMETIC

- Arithmetic is performed with operators
  - + for addition
  - - for subtraction
  - * for multiplication
  - / for division
- • Example: storing a product in the variable
- total_weight
- total_weight = one_weight * number_of_bars;

What is ROOT?

- In HEP, we collect a large amount of data
  - average size of one event (raw) can be greater than a megabyte and we record data from the LHC experiments at a few hundred per second
  - even if we wanted to cannot represent all of that information for all of those events
  - need a way to consolidate and extract the most important pieces of that data out and relate them to basic quantities about the universe
  - We do this by extracting the most relevant information out of each event first and then further compiling information over a large number of events

What is ROOT?

- ROOT is an object oriented (C++) data analysis framework
- It is a series of classes which can be used to represent data and manipulate it for analysis
  - many `native' data types that are common for HEP analysis (histograms, graphs, Lorentz vectors, fitting, probability distributions, etc)
  - framework is supported by CERN and Fermilab - many users contribute libraries for common applications
- Today we will start with the basics of what we want to use it for and how to do simple commands.
- Note that you cannot learn ROOT just by studying lecture notes or reading the manual - you need to have practice and do it yourself
  - Corollary - it is fine to work together and discuss BUT don't just copy somebody else's code (either another student or what you find on the internet). You will learn much less this way...

WAYS OF USING ROOT

- Root can be used in one of two ways
  - `interactively' or 'interpretively' in which ROOT root interprets each line as you type it or as it reads it in from a text file
    - ADVANTAGE - for very simple tasks this is simple and fast
    - DISADVANTAGE - errors get caught at execution time, sometimes leading to catastrophic failure that is hard to understand and difficult to fix
  - `compiled' - where the code is first compiled and optimized in machine language by the compiler and then executed afterwards
    - ADVANTAGE - compiled code is generally faster and easier to debug as the compiler catches much problematic code and can also pinpoint the exact location of something that leads to a crash

CINT

- The interactive version of ROOT is called 'CINT' which just means C++ interpreter
- CINT
  - 'based' on C++ (but not quite standard C++)
  - It is NOT compiled. Sometimes it will do wrong things without warning (eg. convert or compare numeric types without letting you know). This can be dangerous and give wrong results!
  - you may need to restart root more than you like
  - does not distinguish between objects and pointers to objects. This makes it `easier' to call and manipulate objects but again can have surprising results

How to start it

- From the unix prompt you can type`root'
- this will take you to the interactive version of root
  - typing .q will quit the program
- Two ways to work interactively with interpreted root
  - type directly into the command prompt (basically only useful for one liners)
  - create a `macro' using a text editor

```
% root
   ******************************************
   *                                        *
   *        W E L C O M E  to  R O O T      *
   *                                        *
   *   Version   5.20/00      24 June 2007   *
   *                                        *
   *  You are welcome to visit our Web site *
   *          http://root.cern.ch           *
   *                                        *
   ******************************************

ROOT 5.20/00 (trunk@24525, Jun 25 2008, 12:52:00 on linux)

CINT/ROOT C/C++ Interpreter version 5.16.29, June 08, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

```
root [0] float myVar = 7
root [1] myVar
(float)7.0000000000000000e+00
root [2]
```

```
float x[5] = {10.0, 20.0, 30.0, 40.0, 50.0};
float y[5] = { 1.1,  1.2,  1.3,  1.4,  1.5};
float dy[5] ={ 0.1,  0.1,  0.1,  0.1,  0.1};
```

ROOT MACRO

- A macro is just a series of lines of code that are executed consecutively. Rather than typing them in all at the command line which can be painful if you make a mistake you can save it in a file and then run it.

- For example the following text file was saved as loop.C and then executed

  - note that everything that is executed is within { }

  - further note that as written this would generate compilation error from any standard C++ compiler (why?)

contents of the file

```
{
cout<<" a loop to count from 0 to 9"<<endl;
for(int i= 0; i< 10; ++i) {
        cout<<"i is now "<<i<<endl;
}

}
```

```
ob-d-128-141-134-240:~ kmblack$ root loop.C
```

```
root [0]
Processing loop.C...
 a loop to count from 0 to 9
i is now 0
i is now 1
i is now 2
i is now 3
i is now 4
i is now 5
i is now 6
i is now 7
i is now 8
i is now 9
root [1]
```

SOME ROOT BASICS

- Plotting a function
- Working with histograms
- Working with multiple plots
- Saving your work

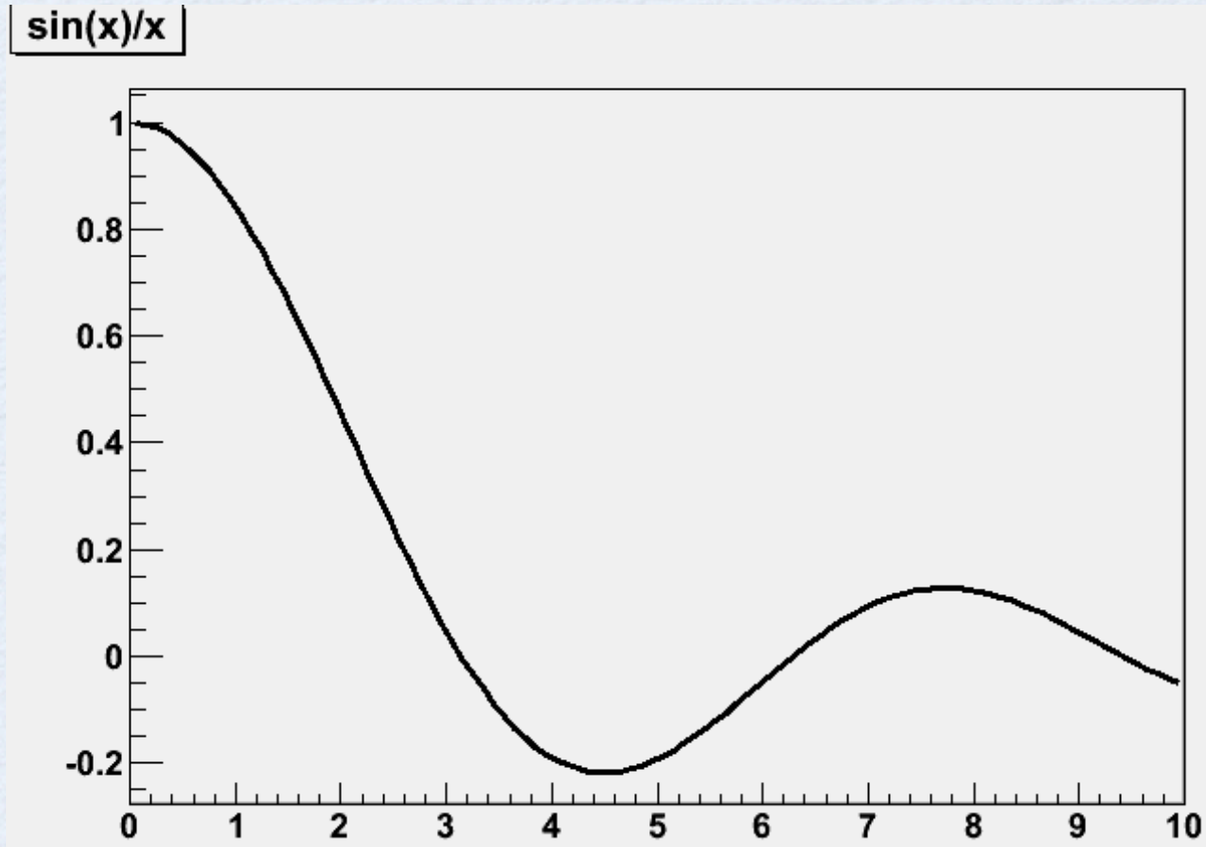
1-D FUNCTIONS

```
root [1] TF1 f1("func1","sin(x)/x",0,10)
```

- The root class for functions is “TF1” (Everything in root is a TSomething - I don’t know why)
- Above is an example of the constructor of the function. What does this mean though:
  - TF1 - we are constructing a C++ object of the type TF1 (think of int myInt =7) - this just tells us what sort of object we want
  - “func1” is the title of the function
  - “sin(x)/x” is the functional form
  - 0 is the lower range
  - 10 is the upper range

FUN WITH FUNCTIONS

- f1.Draw()
  - Draws the function - note that we specified what range x
  - We did not however specify the dependent variable. By default root choses that to show the full range.
  - There are many things we can change on the plot
- http://root.cern.ch/root/html/TF1.html
  - full list - we will show a few examples and you can then figure it out



note the inexplicable
default off white
background. This is ok
on slides but is very visible on white paper

C++ ASIDE

- On the previous slides I referred to TF1 as being a `class'
- This has a very particular meaning in C++.
- This gets to the heart of what we call Object Oriented Computing
- In particular we say that we instantiate an **object** of the **class** type TF1 which is called f1

WHAT IS OBJECTED ORIENTED PROGRAMING

An object is like a black box.
The internal details are hidden.

- Identifying objects and assigning responsibilities to these objects.
- Objects communicate to other objects by sending messages.
- Messages are received by the methods of an object

What is an Object

- Tangible Things as a car, printer, ...
- Roles as employee, boss, ...
- Incidents as flight, overflow, ...
- Interactions as contract, sale, ...
- Specifications as color, shape, ...
- In HEP we could mean things like - a particle, a function, a detector element, ...

WHY DO WE CARE

- Often in programing for HEP we are describing concepts which we have specific properties and that we will use over and over again
  - Events - each collision that we record
  - Particle - Individual particle (sometimes of different types) in the event
  - 4-vector
    - A drift tube in the detector - one of many thousand
- Object oriented programing tries to represent these concepts in to an abstractions called **classes**
- Each class is suppose to represent that idea and we build in a set of functions and data for each class

WHY DO WE DO THIS

- Modularity - large software projects can be split up in smaller pieces.
- Reusability - Programs can be assembled from pre-written software components.
- Extensibility - New software components can be written or developed from existing ones
- Basically - it is one approach to trying to organize the way we think about writing code
- Note - for very simple things it is not at all obvious why you go through all this trouble. If you wanted to make one calculation once it is far from obvious you would do this!

EXAMPLE

```
#include<string>
#include<iostream>
class Person{
   char name[20];
   int yearOfBirth;
public:
   void displayDetails() {
         cout << name << " born in "
               << yearOfBirth << endl;
         }
    //...
};
```

private data

public processes

BUT WAIT...

```
root [1] TF1 f1("func1","sin(x)/x",0,10)
```

- We say we are 'constructing' an object f1 which is of class type TF1
- We pass it the name, functional form, and range and these all become **private data members** of that object
- When we write things like f1.Draw() what we are in fact doing is calling the **member function**
- We say this is an example of **encapsulation.** From the purely coding point of view we don't know all the details of what Draw() actually does. We didn't have to write the code for it. We just use it
- This is the heart of the object oriented programming. The idea is to
  - make code more organized
  - allow us to develop complicated code which the user can call relatively simply

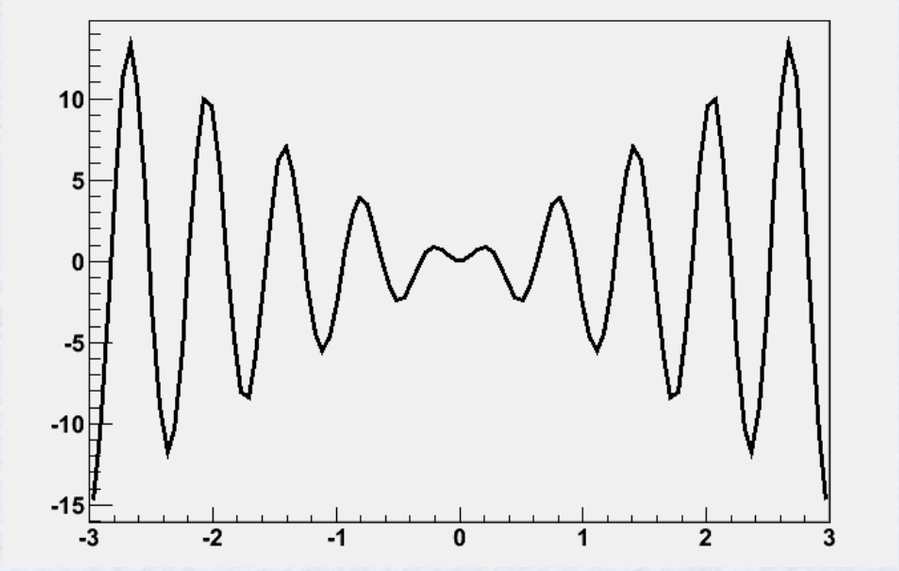
NOTE OF CAUTION

- The concept of information hiding or encapsulation is a very useful one for the purposes of writing code
- however, often this is a bit antithetical to what you want to do as a physicist - i.e you want to know all the details!
- You can always look through the source code to find out what is actually done. Though sometime this can be a real effort!

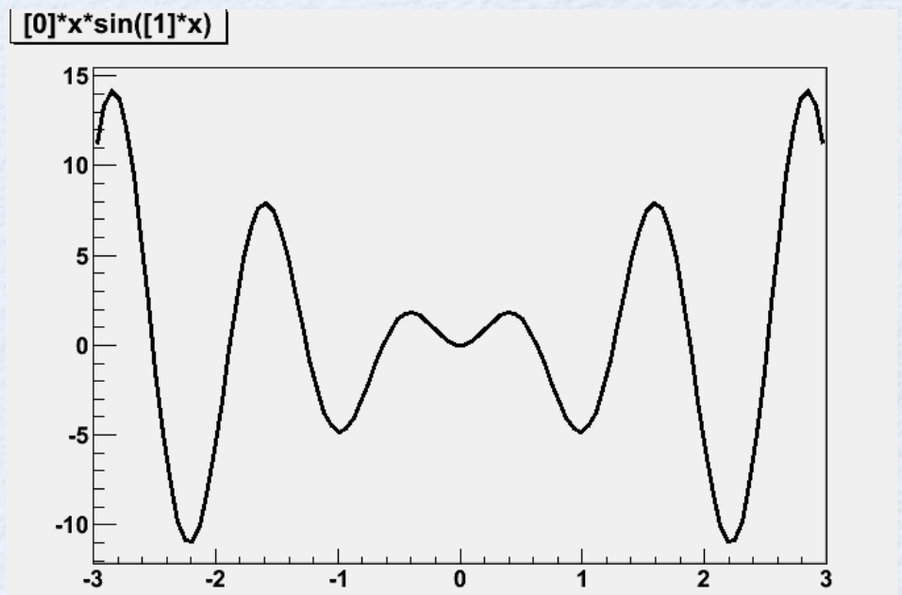
Functions with parameters

- Many times we need to have parameters
  - For example if we know the general shape of the distribution but need to extract an amplitude or a frequency
- ROOT allows us to do that by having parameters which we are free to adjust (or fit for)
- It is very common that we have some spectrum which has been measured and have an expectation or a guess about the shape which we then want to extract
- we can change the frequency and redraw

```
root [26] TF1 *fa = new TF1("fa","[0]*x*sin([1]*x)",-3,3);
root [27]  fa->SetParameter(0,5);
root [28]  fa->SetParameter(1,10);
root [29] fa->Draw()
```




```
 fa->SetParameter(1,5);
fa->Draw()
```



CLOSER LOOK

```
root [26] TF1 *fa = new TF1("fa","[0]*x*sin([1]*x)",-3,3);
root [27]  fa->SetParameter(0,5);
root [28]  fa->SetParameter(1,10);
root [29] fa->Draw()
```

- Something is different about the code syntax in the first line

CLOSER LOOK

```
root [26] TF1 *fa = new TF1("fa","[0]*x*sin([1]*x)",-3,3);
root [27]  fa->SetParameter(0,5);
root [28]  fa->SetParameter(1,10);
root [29] fa->Draw()
```

- Something is different about the code syntax in the first line
- Look at the structure
  - it contains a C++ keyword `new'
  - rather than creating an object it creates a pointer to an object
  - we say it `allocates' a block of memory using the new opperator

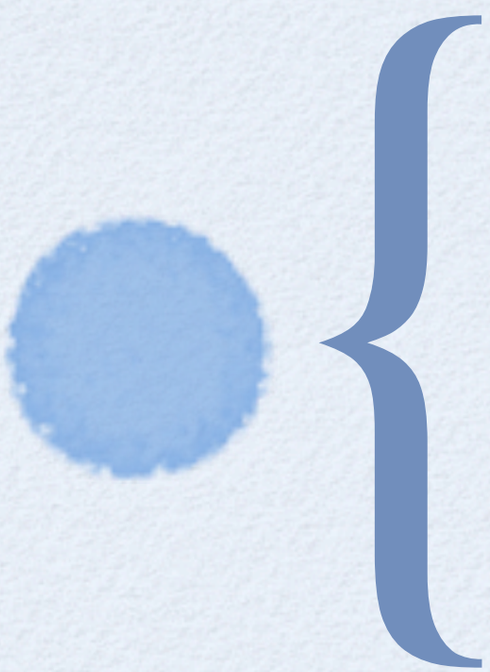
the object that we are requesting is of type TF1

TF1 *fa = new TF1(...)

pointer to an object of type TF1

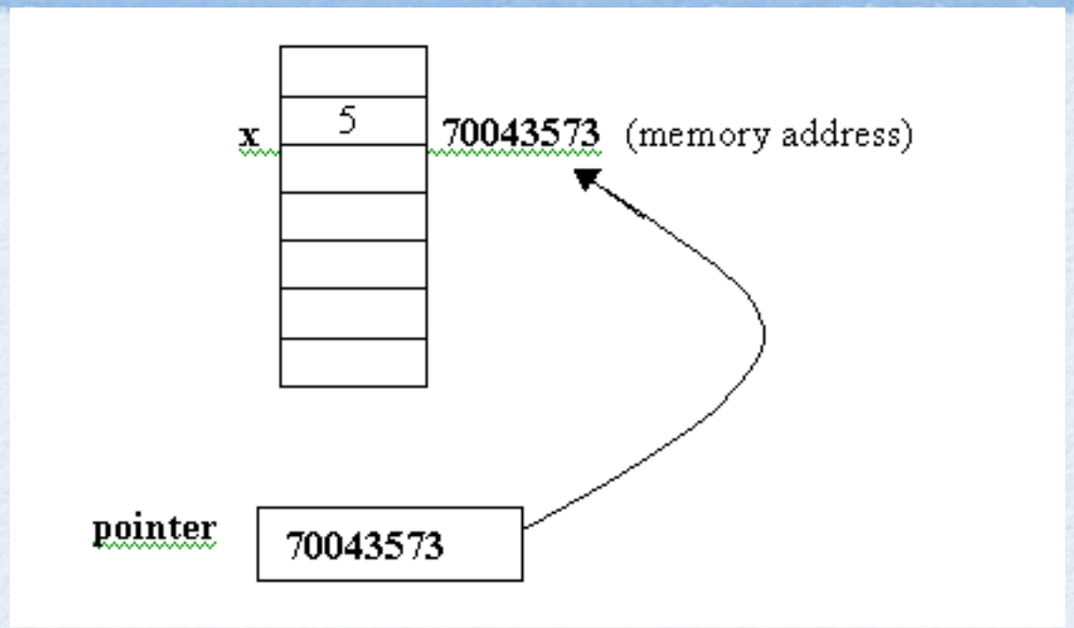
C++ keyword

A PARENTHETICAL COMMENT



REMINDER OF POINTERS

- Computers operate by preforming binary operations on numbers
  - you will need to take a digital electronics class to understand how this is done in detail
- When you create an `object' in memory this refers to a particular space in addressable memory where the CPU can
- this means that you can reference an object rather than access it directly



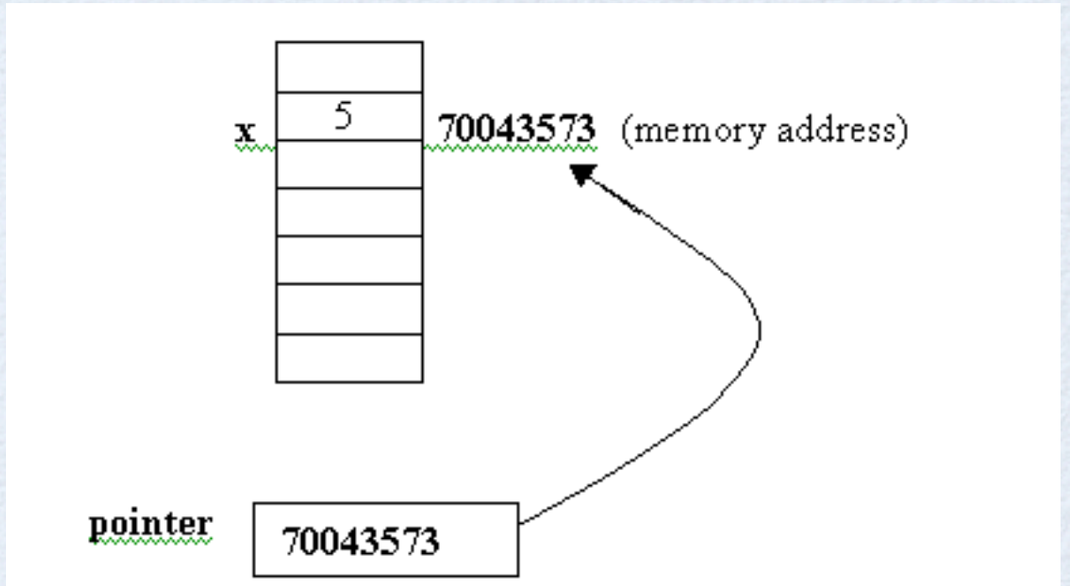

```
{
   int x = 5;
   int *pointer;
   pointer = &x;    // &x refers to the memory address of x
   cout << "x = " << x << endl;
   cout << "pointer = " << pointer << endl;
   cout << "*pointer = " << *pointer << endl;
}
```

OK FINE - BUT WHY?

- In the context of the pointer to the integer you might wonder why we do this at all?

- In the case of this example it seems a bit silly.

- Why not just manipulate the object itself?

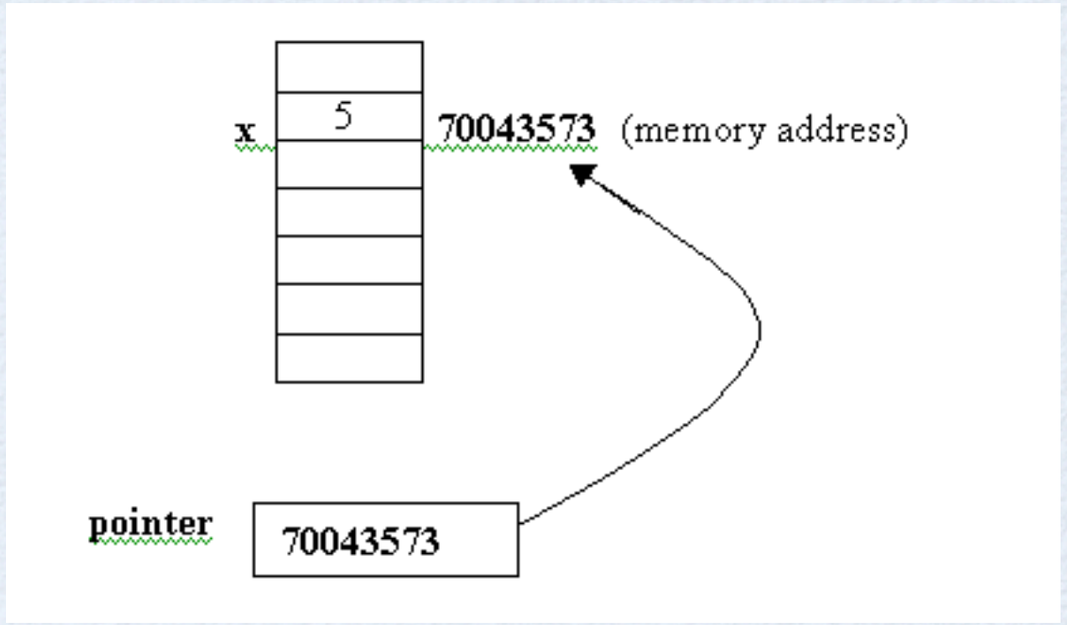


OK FINE - BUT WHY?

- The reason is that C++ objects can be very complicated and take up a large amount of memory - not just one space

- imagine that the object is not a simple integer but rather the sin function or something more complicated

  - using as a pointer as a reference saves a lot of copying both in memory space and in time



CLASSIC EXAMPLE

Parameter Passing

pass by value

```
int add(int a, int b) {
  return a+b;
}

int a, b, sum;
sum = add(a, b);
```

Make a local copy of `a` & `b`

pass by reference

```
int add(int *a, int *b) {
  return *a + *b;
}

int a, b, sum;
sum = add(&a, &b);
```

Pass pointers that reference `a` & `b`. Changes made to `a` or `b` will be reflected outside the `add` routine

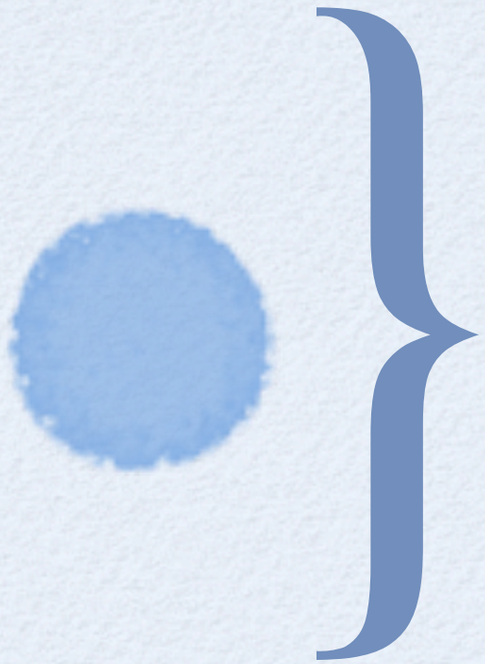
WHY DOES THIS MATTER

- If you have one function that is called only once with small objects making a local copy won't waste much space.
- If you have very complicated objects that take a large amount of memory and you call the function 100,000 times before you can reset the memory - it can be very slow
- Some computer languages handle all memory management `automatically'
  - less freedom in how your code executes
  - but allowing the programer to allocate blocks of memory by him/herself alows more control - BUT it is very easy create leaks!

What is a memory leak?

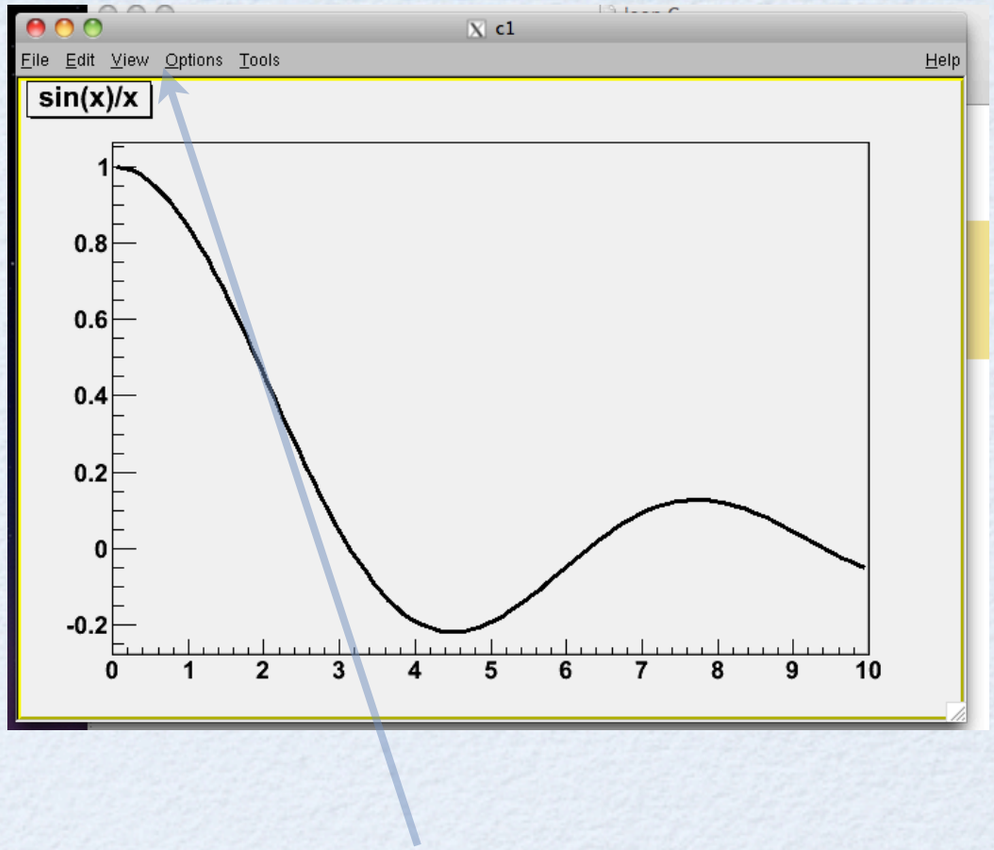
- The C++ `new' operator allocates memory to a particular object and freezes it from other uses (so that later in the program you can get back this information)
- However, until you use the keyword `delete' it remains frozen in memory and cannot be used by the CPU for any other purpose
  - This means there is less space available for other computations and eventually your program may crash
  - Imagine that there are only two slots in memory and you use them both. Now you want to create a 3rd integer? There is no where to put it and the program stops execution
  - Generally - it also slows down computation as the computer uses (virtual memory -i.e. temporarily uses the hard disk as memory which takes much longer to read and write to)

END COMMENT



BACK TO FUNCTIONS

- There are two ways to make changes to the style of plot
  - interactively through the menus (easiest for simple things)
  - directly in through the code (generally better to do it this way when you are making a lot of plots and you almost NEVER make a plot only once!)

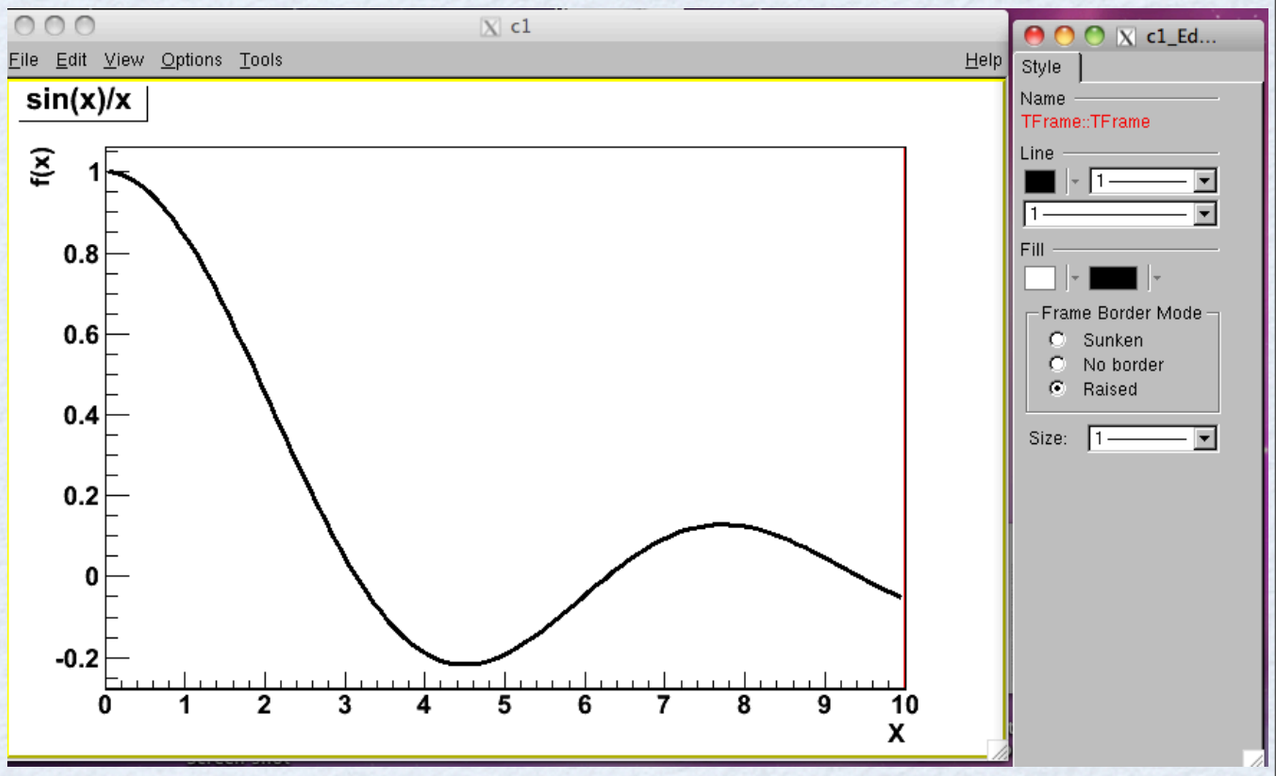


use the menu with the mouse!

POINT AND CLICK

- If you right click on various objects on the canvas you can access their properties through a GUI and add axis labels, changes colors, line width etc

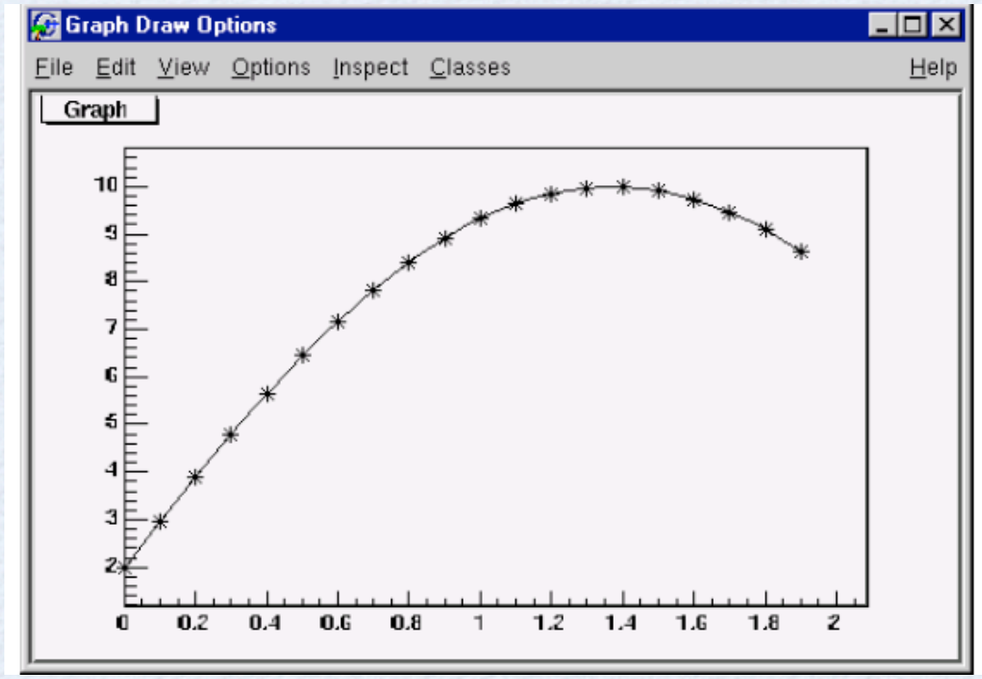
- The GUI is actually accessing the underlying ROOT/C++ objects and modifying their properties. Every thing you see on the screen shot is a class (function, title, label, canvas, and the GUI itself!)



TGRAPH

```
{
  Int_t n = 20;
  Double_t x[n], y[n];
  for (Int_t i=0;i<n;i++) {
    x[i] = i*0.1;
    y[i] = 10*sin(x[i]+0.2);
  }
  // create graph
  TGraph *gr  = new TGraph(n,x,y);
  TCanvas *c1 = new TCanvas("c1","Graph Draw Options",200,10,600,400);
  // draw the graph with axis, contineous line, and put a * at each point
  gr->Draw("AC*");
}
```

- Besides functions we can also plot data in several ways. The simplest is a "TGraph"
- This is just a collection of points which can be connected
- Various drawing and connecting options



CLOSER LOOK

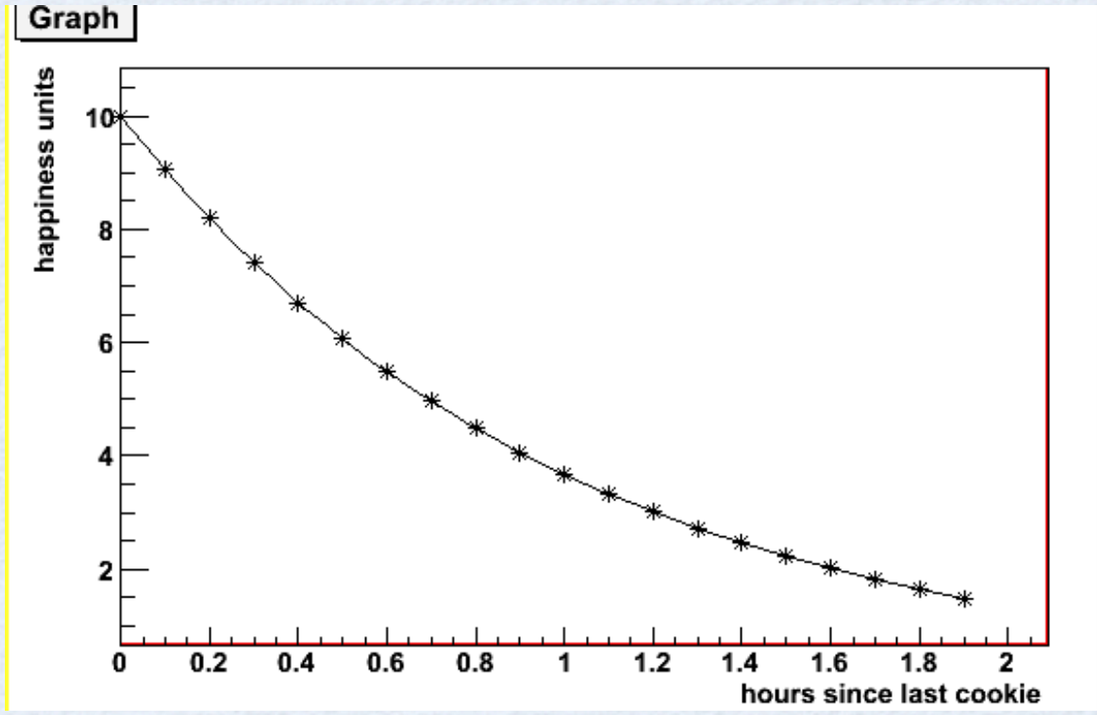
- Create two arrays of variables of type double
- Fill them with some values
- Create a graph with the number of points and the two variable arrays you want plotted against each other
- TCanvas is just ROOTs name for the little window that pops up and where you can place one or multiple objects to be displayed

```
{
  Int_t n = 20;
  Double_t x[n], y[n];
  for (Int_t i=0;i<n;i++) {
    x[i] = i*0.1;
    y[i] = 10*sin(x[i]+0.2);
  }
  // create graph
  TGraph *gr  = new TGraph(n,x,y);
  TCanvas *c1 = new TCanvas("c1","Graph Draw Options",200,10,600,400);
  // draw the graph with axis, contineous line, and put a * at each point
  gr->Draw("AC*");
}
```

Note that this time we explicitly created a TCanvas for the graph to be shown on

How to title an axis from code

- From the pointer to the TGraph you can access pointers to the axis and set titles
  - gr->*GetXaxis*()->*SetTitle*("hours since last cookie");
  - gr->*GetYaxis*()->*SetTitle*("happiness units");
- Everything is an object in root! The “GetXaxis()” function returns a pointer an object of type “TAxis”
- Then you set the properties of that particular TAxis

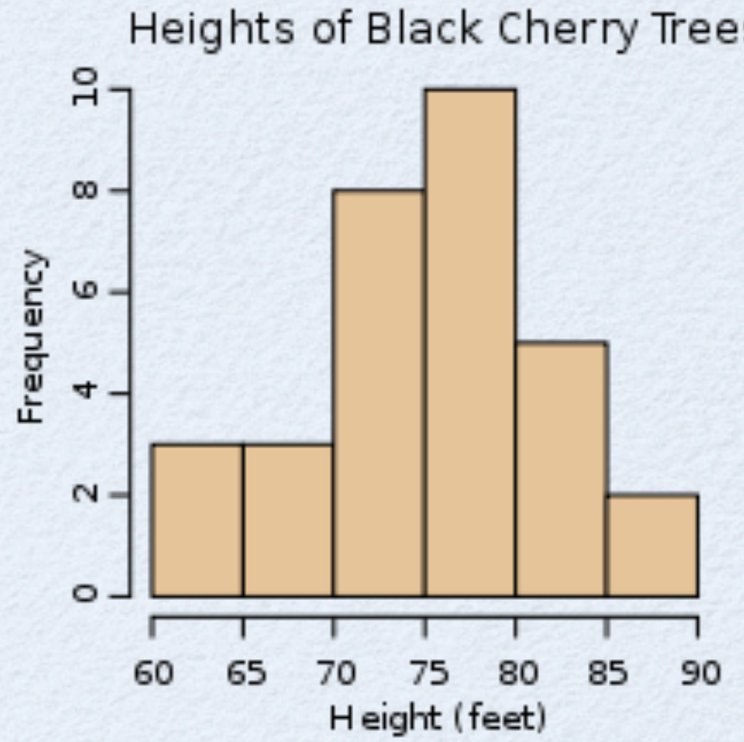


ROOT CLASS INDEX

- There are so many libraries, options, classes we can't possibly cover them all.
- The `ROOT class index
  - http://root.cern.ch/root/html/ClassIndex.html
- Contains every class in root - essentially all the source code in an interactive and online browser
- This in combination with the ROOT manual are the best way to figure out how to

HISTOGRAMS

- Often in HEP we talk about something called "events"
  - For example a proton-proton collision at CMS or ATLAS
  - A neutrino interaction in the water of Super K
- Typically (but perhaps not always) these happen over and over again
  - By analyzing many events together we can measure a quantity related to the probability distribution
- Simplest version - frequency on vertical axis and some physical quantity on the the x-axis



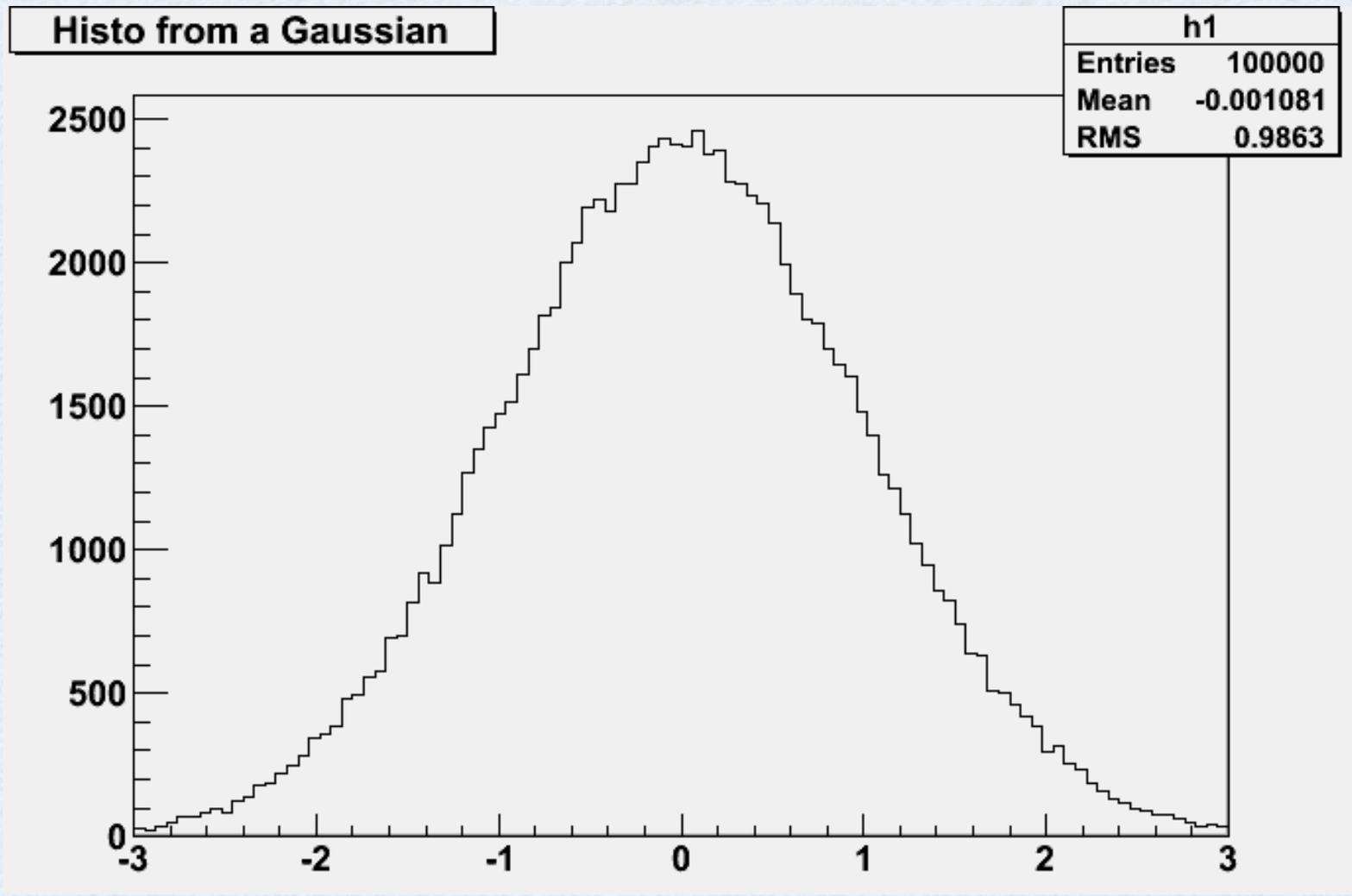
WHY HISTOGRAM?

- In HEP we often collect data as a series of events
- Too much information to collate as a giant list of 4 - vectors
- helps us summarize and is a sample of the full distribution with a finite number of events

TH1F

```
root[] TH1F h1("h1","Histo from a Gaussian",100,-3,3);
root[] h1.FillRandom("gaus",10000);
```

h1.Draw()



BEST WAY TO LEARN

- Learn by doing!
- http://root.cern.ch/root/html/tutorials/
- Large number of simple (and some complicated) pieces of root macros
- Run some of these and try to understand the code
- Try making simple modifications and see what happens

Next week

- Reminder first homework due
- How we collect data and process it in HEP experiments
- More topics in root (data structures, files, basic calculations)
- email me with questions!