Basic Computing Techniques

Road Map

- Getting started with ROOT
- C++ Review
- Getting started with data presentation
There are several programs available

- ROOT <-- What we'll use
- IDL
- Origin
- PAW
- &c

All are pretty good, but you need to be careful and understand what each one is doing.

Remember, you are responsible for your results. If there is a bug in a program and you get the wrong answer, it is still your fault.

Most of the semester will be agnostic about which analysis program is used, but for today we'll concentrate on ROOT.
“Required” Analysis Features

- Basic Graphics and Plotting
- Simple 3D Graphics
- Numeric Function Optimization

Interactive Control Language (macros)
Interface to some programming language.
ROOT: A Data Analysis Frame

- Available from <http://root.cern.ch/>
  - Install the latest production version (5.14/00)
  - Linux: There are rpms available for installation
  - Windows: Compiled executables are available, require a C++ compiler
    - Works with the few command line version.
- ROOT will be installed on the Math SINC site linux machines.
  - Site becomes available next week.
- My basic instructions will be for ROOT on linux
  - Except for command line, works the same on windows.
Starting and Stopping ROOT

Command to run ROOT

```
mcgrew@boxer:~$ root
*******************************************
*                                             *
*        W E L C O M E to R O O T             *
*                                             *
*   Version  5.12/00f   23 October 2006       *
*                                             *
*  You are welcome to visit our Web site      *
*    http://root.cern.ch                     *
*                                             *
*******************************************
```

The ROOT version

FreeType Engine v2.1.9 used to render TrueType fonts.
Compiled on 21 January 2007 for linuxdeb with thread support.

The C++ interpreter version

```
CINT/ROOT C/C++ Interpreter version 5.16.13, June 8, 2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
```

C. McGrew's Style Configuration
```
root [0] .q
mcgrew@boxer:~$
```

“.q” exits ROOT. In root the command line takes a C++ statement, or a “control” which starts with “.”
Entering C++ Commands

```
mcgrew@boxer:~$ root
 ************************************************
 *                                                *
 *        W E L C O M E  to  R O O T            *
 *                                                *
 *   Version  5.12/00f   23 October 2006       *
 *                                                *
 *  You are welcome to visit our Web site        *
 *    http://root.cern.ch                       *
 *                                                *
 ************************************************

FreeType Engine v2.1.9 used to render TrueType fonts.
Compiled on 21 January 2007 for linuxdeb with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.13, June 8, 2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.

C. McGrew's Style Configuration
root [0] for (int i = 0; i<3; ++i) {
  end with '}', '@':abort > std::cout << "hello world"
  end with '}', '@':abort > std::endl;
  end with '}', '@':abort >
} hello world
hello world
hello world
root [1] .q
mcgrew@boxer:~$
```

Command to run ROOT

C++ “for loop”

Output from for loop
Simple “Macro” Files

- C++ from the command line is “a bit” troublesome, so most of the time you will use macro files.
- Create macro files with your favorite editor.
- End the file name with “.C” (the capitalization matters).
  - e.g. “hello.C”

```c++
// An example “hello world”
for (int i = 0; i<3; ++i) {
    std::cout << “hello world”
              << std::endl;
}
```

- Macros start with a “{”
- Macros end with a “}”

Run macro in root

```
mcgrew@boxer:macros$ root -n -l
root [0] .x hello.C
hello world
hello world
hello world
root [1] .q
mcgrew@boxer:macros$
```

- Options to make it less verbose
  - “.x” executes the “hello.C” macro
  - “.q” exits ROOT.

- Macro files are “just plain” C++ code
C++ output is done using “streams” which are just places to put output

- Output to the screen is done to “std::cout”
  - std::cout << “output”
  - This prints “output” on the screen.
  - Notice the “<<” operator says “Put 'output' into 'std::cout'”

- Output to a file is done using a “ofstream”
  - That's read “output file stream”
  - Here is the previous “hello.C” macro modified to write to the file “hello.txt”

```cpp
{ // A simple hello world example.
    std::ofstream file("hello.txt");
    for (int i = 0; i < 3; ++i) {
        file << "hello world"
            << std::endl;
    }
    file.close();
}
```

- Most variables can be printed just by putting them into the output stream.

```cpp
{ int i = 3;
    std::cout << “The value of i is “ << i
            << std::endl;
}
```
C++ Review: Declaring Variables

C++ Variables must be declared before they are used.
- Integer are declared with “int”
- Floating point numbers are declared with “double”
- Strings are declared with “std::string”

```c++
{
    int i = 2;
    double s = 3.14159;
    std::string pi = "apple";
    std::cout << "i is " << i << std::endl
               << "s is " << s << std::endl
               << "pi is " << pi << std::endl;
}
```
C++ Review: Input Streams

- C++ input is done using streams which are places to get input from.
- Input from the keyboard is done with `std::cin`:
  - `std::cout >> i;`
  - The operator “>>” puts the input into “i”
- Input from a file is done using an “std::ifstream”:
  - This is read “input file stream”

```cpp
{
    std::ifstream input("datafile.txt");
    int i;
    for (;;) {
        input >> i;
        if (!input.good()) break;
        std::cout << i;
    }
    input.close();
}
```
C++ Review: Arrays

- Arrays are declared by adding “[]” with a count
  - `int myArray[3]; // An array of three integers`
- Arrays are indexed from zero
- The macro

```cpp
{  
    int myArray[3];  
    myArray[0] = 1;  
    myArray[1] = 2;  
    myArray[2] = 3;  
    for (int i=0; i<3; ++i) {  
        std::cout << " " << myArray[i];  
    }  
    std::cout << endl;  
}
```

- prints “1 2 3”
C++ Review: Pointers

- Most variables just contain a value
  - An “int” variable contains an integer value
  - A “double” variable contains an floating point value
- But, sometimes (quite often) you want to hold the **address** of a value
  - This is the definition of a pointer.
- Pointers are declared using the “*”
  - `int* pointerToInteger;` // A variable holding the address of an integer
- The value “pointed to” by the variable is access using “*”
  - `*pointerToInteger = 5;` // Put “5” at the address pointed
  - `int theInteger = *pointerToInteger;`
  - Also “->” -- more on that later when I talk about classes.
- To use a pointer, you have to assign it an address
  - `int theInt = 5;`
  - `int* pointer = &theInt;` // pointer holds theInt address
  - `int* pointer = new int;` // Allocate new space for the value.
  - If you allocate space you must delete it with “delete pointer;”
C++ Review: Conditional Statements

Conditional statements are the if-then-else clauses

```cpp
if (condition) {
    // Code executed if condition is true
}
else {
    // Code executed if condition is false
}
```

Logical Conditions

```cpp
if (1<2) std::cout << "true";
if (1>2) std::cout << "false";
if (1==2) std::cout << "false";
if (1<2) std::cout << "true";
if (1>2) std::cout << "false";
if ((1<2) && (1>0)) std::cout << "false";
if ((1<2) || (1>0)) std::cout << "true";
```
C++ Review: Loops

- C++ provides the “for loop” as an all-purpose looping construct
  - for (initial conditions; execute while true; increment) { code }

```cpp
for (int count = 0;
     count < 5;
     ++count) {
    std::cout << "Count is " << count
              << std::endl;
}
```

- You can break out of a loop (terminate early) using the “break” statement

```
for (int count = 0;
     count < 5;
     ++count) {
    if (count == 3) {
        break;
    }
    std::cout << "Count is " << count
              << std::endl;
}
```
C++ Review: Functions

- C++ “subroutines” are called functions
- May (or may not) return a value

```cpp
void voidFunction(int aValue) {
    std::cout << "value is " << aValue;
}

int integerFunction(int aValue) {
    std::cout << "input is " << aValue;
    int myValue = aValue + 1;
    std::cout << "output is " << myValue;
    return myValue;
}
```
ROOT Tutorial: Graphs

- ROOT graphs are ideal for drawing Y vs X graphs (dependent vs. independent variables)
  - By convention, the independent variable goes on the horizontal axis.
- Data can be automatically read from a file
  - You can control the reading format, but the default works for most files
- ROOT is heavily object oriented
  - Provides a (huge) library of classes that do (almost) everything needed for an analysis
  - ROOT classes all start with a capital “T”, so a graph is created using the “TGraph” object.
    - Declare a pointer to a TGraph
    - Allocate a new TGraph object
      - Data is added to the TGraph during the constructor call (see next slide)
    - Draw the TGraph
      - After drawing, the picture can be saved to a file.
- Most work is done by writing macro files and executing them inside of ROOT
TGraph Example

Plot data from an Ohms law experiment

<table>
<thead>
<tr>
<th>Voltage (V)</th>
<th>Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ± 0.02</td>
<td>0.37 ± 0.02</td>
</tr>
<tr>
<td>2 ± 0.04</td>
<td>0.71 ± 0.04</td>
</tr>
<tr>
<td>4 ± 0.08</td>
<td>1.4 ± 0.07</td>
</tr>
<tr>
<td>5 ± 0.10</td>
<td>1.9 ± 0.10</td>
</tr>
<tr>
<td>7 ± 0.14</td>
<td>2.41 ± 0.12</td>
</tr>
<tr>
<td>10 ± 0.20</td>
<td>3.39 ± 0.17</td>
</tr>
</tbody>
</table>

The result will look like this
A ROOT Macro to Make a Graph

**ohmsLaw.C**

```c
{ TGraphErrors* g = new TGraphErrors("ohmsLaw.dat");
g->SetTitle("Current vs Voltage");
g->GetHistogram()->SetXTitle("Voltage (V)");
g->GetHistogram()->SetYTitle("Current (mA)");
g->Draw("AP");
gPad->Print("ohmsLaw.png");
}
```

- Allocate a new graph
- Set the title
- Label the X axis
- Label the Y axis
- Draw the graph on the “canvas”
- Save the picture to a file

Most of the work is done by calling the class methods

> “->” is read “points to”

The window where ROOT draws a picture is called a “Canvas”

> ROOT creates a canvas when it's needed

> A pointer to the current canvas is saved in the global variable “gPad”
**C++ Review: Using Classes**

- Classes are the C++ workhorse. We will be using, but probably not writing classes
  - A class creates a new variable type that ‘is used just like a regular type”

- What we need to know
  - A class has internal **data** that saves the state of an object
    -> The data is not shared with other objects of the same class
  - A class has **methods** which are like functions
    -> Methods act on the class data

- In ROOT, we will usually use pointers to class objects.
  - Example – streams are actually class objects

```cpp
{  
  std::ofstream *file  
  = new std::ofstream("myFile.txt");  
  (*file) << "hello worlds" << std::endl;  
  file->close();  
  delete file;  
}
```

- Declare a pointer to a file
  - And assign an address of a new file object
- Reference the object using the “*” operator
- Reference the object using the “->” operator
- Delete the object referenced by “file”
ROOT Tutorial: Creating Histograms

- ROOT histograms are a type of graph that has been specialized to show the frequency of measurements.
  - By convention, the measurement goes on the X axis and the number of measurements at a particular value goes on the Y axis.
- Histograms are filled by adding one entry at a time
- Data is not automatically read in from a file
  - However, with histograms you often do calculations with the data before “filling” so this isn't really a limitation.
- ROOT provides several histogram classes
  - We'll mostly use “TH1F” which is a one dimensional histogram of floating point measurements
  - Later we'll use “TH2F” for two dimensional histograms.
A ROOT Macro to Make a Histogram

Create a histogram from data in a file.

```cpp
{ 
  TH1F* h
    = new TH1F("mudkTime",
                "Muon Decay Times",
                50,
                0.0,10000.0);

  h->SetXTitle("Decay Times (ns)");
  h->SetYTitle("Decays per 200 ns");
  std::ifstream input("mudkTime.dat");
  double time;
  for (;;) {
    input >> time;
    if (!input.good()) break;
    time *= 1000;
    h->Fill(time);
  }
  input.close();

  h->Draw("E");
  gPad->Print("mudkTime.png");
}
```

Create a histogram named “mudkTime”
With a title “Muon Decay Times”
And 50 bins
ranging between 0.0 and 10000.0

Set the X and Y axis labels.

Read the data file and “Fill” the histogram.

Draw the histogram on the canvas and
save the picture to a file.
Random Number Generation

- We're going to need a lot of data in this class
  - To prove that statistics works
    - We need to know the “true” distributions
    - We need to repeat the experiments thousands of times
  - The budget for PHY310 lab equipment...
    - minimal (as in non-existent)
- Solution:
  - A lot of the time, we will use ‘Fake” data
    - Usually called “Monte Carlo” data
  - Monte Carlo data is designed to simulate measured data
    - The best MC data is indistinguishable from measured data.
- In ROOT we generate random numbers using the gRandom object.
Filling a Histogram with a Gaussian

```cpp
// Make a 1D Gaussian histogram.
TH1F* gaus1D = new TH1F("gaus1D", "One Dimensional Gaussian", 60, -3, 3);
gaus1D->SetXTitle("Sigma");
gaus1D->SetYTitle("Counts per 0.1 sigma");
for (int i=0; i<100000; ++i) {
    double x = gRandom->Gaus(0,1);
    gaus1D->Fill(x);
}
gaus1D->Draw("E");
gPad->Print("gaus1D.png");
```

- Create the histogram and label the X and Y axis
- Fill the histogram with 100,000 random numbers from a Gaussian distribution
- Draw the histogram and save the picture to a file

Other Useful Random Distributions

- Uniform between [0.0, 1.0) – gRandom->Uniform()
- Poissonian – gRandom->Pois(3.3)
  - This has a mean of 3.3
- Exponential – gRandom->Exp(2.1)
  - This has a mean of 2.1
Finally

Today we covered most of the technical computer stuff. Anything else we need will be picked up as we go along.

C++ is a good tool to have in your box for later.
- You might want pick up a book on C++
  - The gold standard is Stroustrup's “The C++ Programming Language”
  - If you have Linux, it comes with an excellent C++ compiler (g++)
- If you use Windows, there is a free version of Visual C++
  - I think the current version is something 2005.

Starting next week we'll talk about probability, uncertainty, and what you're really trying to say when you report a measurement.

The End