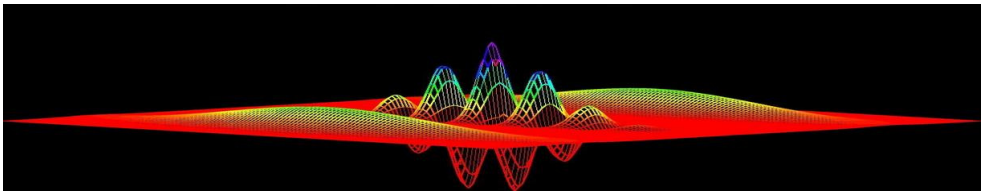
Computational Physics

numerical methods with C++ (and UNIX)

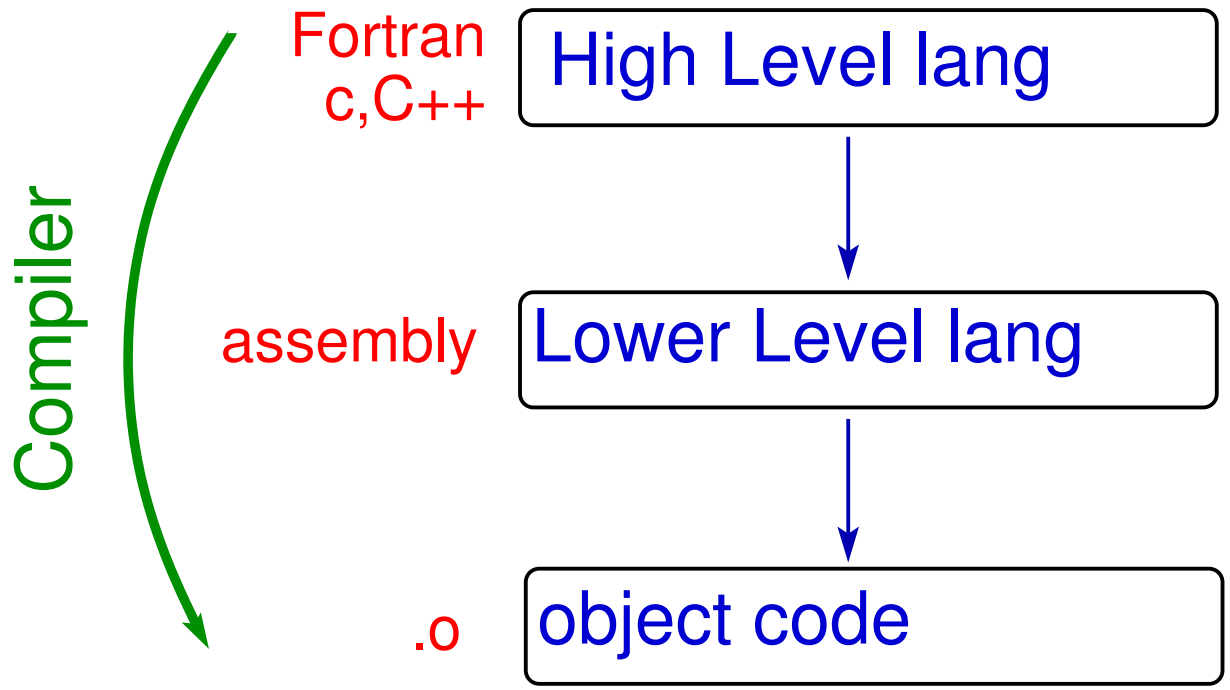


Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: barao@lip.pt

Computer programming

- ✔ Symbolic languages use words (“add”, “move”, ...) instead of operation codes
- ✔ High-level symbolic languages :
  - ► FORTRAN FORmula TRANslator mid 1950’s
  - ► BASIC Beginner’s All-purpose Symbolic Instruction Code mid 1960’s
  - ► PASCAL early 1970’s
  - ► C mid 1970’s
  - ► C++, Java, ... mid 1980’s on
- ✔ C and C++ allow the manipulation of bits and bytes and memory addresses (some people tag it as mid-level languages)
- ✔ Other languages like Mathematica, Matlab or Maple : very rapid coding up but...code is interpreted (slower)
- ✔ The lowest level symbolic language is called the *assembly language*
- ✔ The **assembler** program translates the assembly into **machine code (object code)** that will be understood by the CPU

Computer programming



Creating an executable

- ✔ An executable file contains binary code encoding machine-language instructions
- ✔ To create it, we need to start by writing a program in a symbolic language, **the source code**
  - ► use some Unix editor like **pico, gedit, emacs**
- ✔ Next, we produce the object code, by compiling the source code and eventually linking with other pieces of code located in libraries or being compiled at the same time
  - ► compilers : **C++** $\rightarrow$ **g++**, **c** $\rightarrow$ **gcc**, **FORTRAN** $\rightarrow$ **gfortran**
  - ► the compiler assigns memory addresses to variables and translates arithmetic and logical operations into machine-language instructions
- ✔ The object code is loaded into the memory (RAM) and it is runned by the CPU (no further need of the compiler)
  - ► the object files are specific to every CPU and are not necessarily portable across different versions of the operating system

ROOT - adding user classes to ROOT

To add user classes to ROOT it is necessary to add two calls to functions that links the classes to the dictionary.
The **ClassDef** and **ClassImp** macros are necessary to link your classes to the dictionary generated by CINT to get access to the RTTI and object I/O features of ROOT. The RTTI system allows you to find out to which class an object belongs, its baseclasses, its datamembers and methods, the method signatures, etc. This information is used to make advanced object browsers and by the automatic documentation generation system. The object I/O system allows you to store and retrieve objects (and arbitrarily complex object structures) from a ROOT database.

- ✔ ClassDef(ClassName, ClassVersionID) ;
- ✔ ClassImp(ClassName) ;
- ✔ A ROOT dictionary is created with the **rootcint** program

```
class A {
 public:
  ClassDef(A);
 private:
};
 // header file includes
 ClassImp(A); //before implementation code
```


ROOT - user interface to ROOT

- ✔ User class methods of general interest can be placed available in a *Computational Physics library* - collaborative work ! The user has to link its C++ program with this library to get access to the class methods there implemented
- ✔ In order to provide an easy user interface to graphics displays using ROOT I made a starting class interface *cFCgraphics class* that provides access to ROOT in a easy way

cFCgraphics.h

```
#ifndef __CFCG__
#define __CFCG__

#include <string>
using namespace std;

class TObject;
class TApplication;

class cFCgraphics {
 public:
  cFCgraphics();
  ~cFCgraphics();
  void AddObject(TObject *obj=NULL );
  void ListObjects();
  void Draw(string sopt=''all'');
  int GetNumberOfListEntries();
 private:
  TApplication *gMyRootApp;
  TList *L; ///< list of user ROOT objects
};
#endif
```

ROOT - user interface to ROOT (cont.)

cFCgraphics.C

```
#include <TROOT.h>
#include <TGFrame.h> //gClient
#include <TCanvas.h>
#include <TPad.h>
#include <TSystem.h>
#include <TList.h>
#include <TApplication.h>
#include <TVirtualX.h>
#include <TF1.h>

#include <iostream>
using namespace std;
#include "cFCgraphics.h"

cFCgraphics::cFCgraphics () {
  L = new TList(); //create TList
  //create application
  int argc = 0;
  char **argv = NULL;
  gMyRootApp = new TApplication(''Comput Phys Application'', &argc, argv);
}

cFCgraphics::~cFCgraphics () {
  delete gMyRootApp;
  delete L;
}
```


ROOT - user interface to ROOT (cont.)

cFCgraphics.C (cont.)

```
void cFCgraphics::AddObject(TObject *obj) {
  if (obj) {
    L->Add(obj);
    cout << Form(''[cFCgraphics::AddObject] %s added.. .(Total nb objects=%d)\n'', obj->GetName(), GetNumberOfListEnt
  }
}

void cFCgraphics::ListObjects() {
  TListIter next(L);
  TObject *to;
  cout << Form(''[cFCgraphics::ListObjects] (Total nb objects=%d)\n'', GetNumberOfListEntries());
  while ((to=next())) {
    cout<< Form('' ---> %s \n'',to->GetName());
  }
  cout<< endl;
}

void cFCgraphics::Draw(string sopt) {
  cout<< Form(''[cFCgraphics::Draw] Drawing objects...[%s]\n'',sopt.c_str());
  //new canvas
  TCanvas *fCanvas = new TCanvas(Form(''FC_canvas_%s'',sopt.c_str()), Form(''Canvas - %s'', sopt.c_str()), 900, 700);
  //all or only one object
  if (sopt == ''all'') {
    // ... get number of list entries
    int count = GetNumberOfListEntries();
    int nlines = count/3;
    int ncolus = count%3;
    if (nlines==0) {
      fCanvas->Divide(ncolus,nlines+1);
    } else {
      fCanvas->Divide(3,nlines+1);
    }
    TListIter next(L);
    TObject *to;
    int n = 0;
```

ROOT - user interface to ROOT (cont.)

The following C++ program adds four ROOT graphics objects to the *cFCgraphics class* to display them.

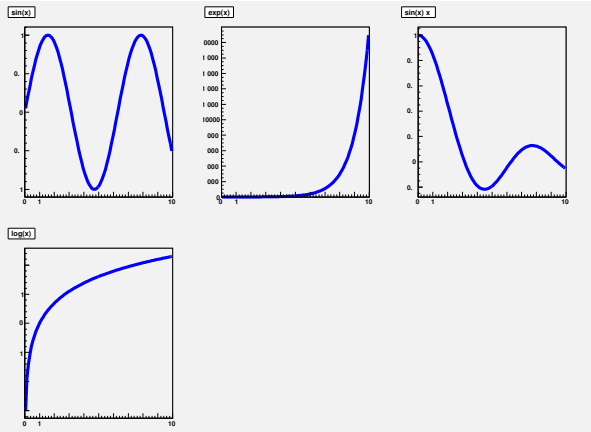
main program : Rgraphics.C

using class cFCgraphics

```
// F Barao (FC, Jul 2014)
// g++ -o Rgraphics.exe \
// Rgraphics.C cFCgraphics.C

#include <cmath> //exp
#include <cstdlib> // for atof(3)
#include <iostream>
using namespace std;
#include <TF1.h> //ROOT TF1
#include "cFCgraphics.h" //my class

int main(int argc, const char* argv[]) {
  cFCgraphics gr;
  TF1 *f1 = new TF1("FCsinx","sin(x)", 0., 10.);
  gr.AddObject(f1);
  gr.AddObject(new TF1("FCexp","exp(x)", 0., 10.));
  gr.AddObject(new TF1("FCsinxx","sin(x)/x", 0., 10.));
  gr.AddObject(new TF1("FClogx","log(x)", 0., 10.));
  gr.ListObjects(); // list user stored objects
  gr.Draw(); // draw user objects
  gr.Print("Rgraphics.eps"); // save objects
  return 0;
}
```



Computational Physics

gnuplot

A data plotting tool

Fernando Barao, Phys Department IST (Lisbon)

gnuplot - start

start

```
linux> gnuplot

        G N U P L O T
        Version 4.6 patchlevel 1     last modified 2012-09-26
        Build System: Linux x86_64

        Copyright (C) 1986-1993, 1998, 2004, 2007-2012
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type ''help FAQ''
        immediate help:   type ''help''  (plot window: hit 'h')

        Terminal type set to 'wxt'

gnuplot>
```

http://www.gnuplot.info

http://www.gnuplotting.org/

gnuplot - function plotter

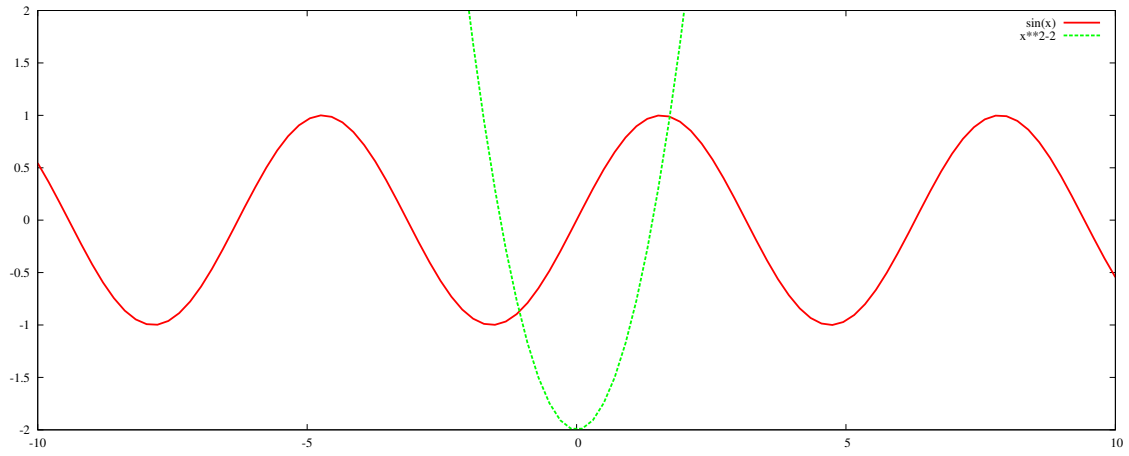
functions

```
gnuplot> set size 2.5,0.7 # landscape plot
gnuplot> set term post portrait color "Times-Roman" 14

gnuplot> plot sin(x), x**2-2  # We are going to draw two functions

gnuplot> plot [][-2:2] sin(x), x**2-2 # define y ranges

gnuplot> plot [][-2:2] sin(x) linewidth 4, x**2-2 linewidth 4 # change linewidth
```



gnuplot - plot saving

There are two ways to save our work in gnuplot :

- ✔ we can save the gnuplot commands used to generate a plot, so that we can regenerate the plot at a later time

save file in gnuplot format

```
gnuplot> set size 2.5,0.7 # landscape plot
gnuplot> plot sin(x), x**2-2  # We are going to draw two functions
gnuplot> plot [][-2:2] sin(x), x**2-2 # define y ranges
gnuplot> plot [][-2:2] sin(x) linewidth 4, x**2-2 linewidth 4 # change linewid
gnuplot> save "MyPlot.gp"
```

- ✔ on next gnuplot session the saved file can be loaded

```
load "graph.gp"
```

- ✔ or we can export the actual graph to a file in one of a variety of supported graphics file formats, so that we can print it or include it in web pages, text documents, or presentations

exporting plots as eps

```
gnuplot> set term postscript eps enhanced color font "Times-Roman" 14
gnuplot> set output "MyPlot.eps"
gnuplot> plot [][-2:2] sin(x), x**2-2 # define y ranges (replot can be used)
gnuplot> set output #back to default screen
```

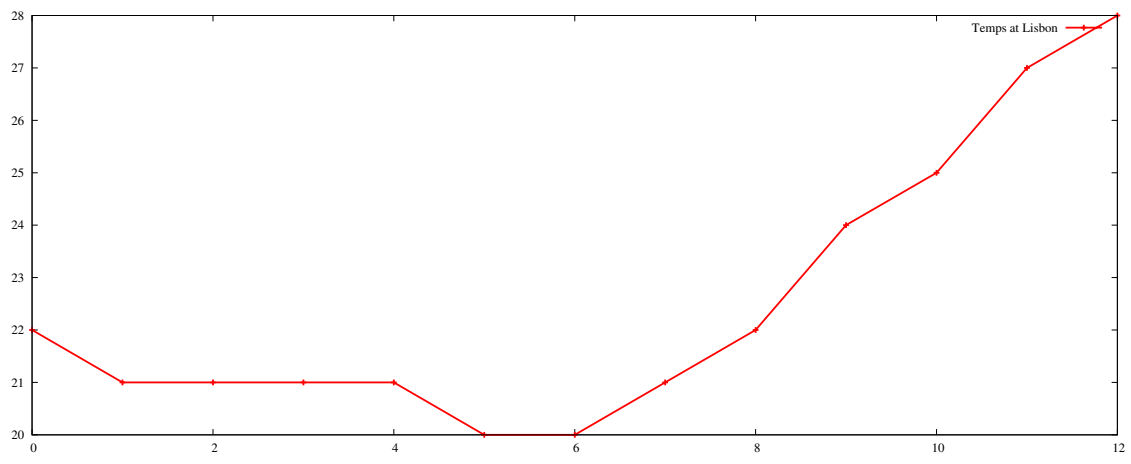

gnuplot - data plotter

data

```
# basic plot
gnuplot> plot "Temps_Lisbon_20Oct2014.txt" using 1:2  # filename : Temps_Lisbon_20Oct2014.txt

# now with lines and title
gnuplot> plot  "Temps_Lisbon_20Oct2014.txt" using 1:2 title "Temps at Lisbon" with lines

# now with lines, points and title
gnuplot> plot  "Temps_Lisbon_20Oct2014.txt" using 1:2 title "Temps at Lisbon" \
              with linespoints linewidth 4
```



Computational Physics
Number representation and Machine precision

Fernando Barao, Phys Department IST (Lisbon)

Introduction

Solving physical problems with a computer

- ✔ numbers and characters representation
  - ▶ binary, decimal and hexadecimal systems
  - ▶ characters
  - ▶ floating point
  - ▶ computation errors

Numbers representation : integer

- ✔ In computers information is stored as a sequence of **0**'s and **1**'s : binary system
- ✔ **Byte** : sequence of 8 bits
  KByte : $2^{10} Bytes = 1024$ Bytes
  MByte : $2^{10} KBytes = 1024$ KBytes
- ✔ A m bits integer number N in binary representation :
  $b_{m-1}\ 2^{m-1} + b_{m-2}\ 2^{m-2} + \cdots + b_0\ 2^0$
- ✔ The sign of the number is stored in one bit (usually the MSB)
  0 = positive
  1 = negative
- ✔ A 32 bits signed integer (4 bytes) uses 31 bits $(0...30)$ for storing the number
  max value of a signed 32bits integer :
  $2^{31} - 1 = \pm 2\ 147\ 483\ 647$

Ex : 417 conversion to binary

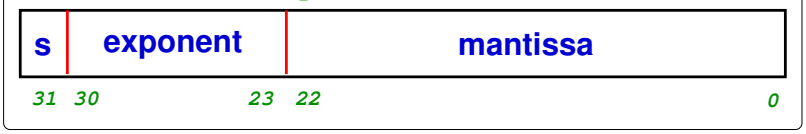
division	remind	coeff
$417/2 = 208$	1	1×2^0
$208/2 = 104$	0	0×2^1
$104/2 = 52$	0	0×2^2
$52/2 = 26$	0	0×2^3
$26/2 = 13$	0	0×2^4
$13/2 = 6$	1	1×2^5
$6/2 = 3$	0	0×2^6
$3/2 = 1$	1	1×2^7
$1/2 = 0$	1	1×2^8

$(417)_{10} = (0...0110100001)_2$

$(-5)_{10} = -(101)_2 = (10...0101)_2$

Numbers representation : reals

32-bits real representation



- ✔ real number has to be converted, **the integral part** and **the decimal part** into a binary
  $\underbrace{634}_{integral} \ . \ \underbrace{28125}_{decimal}$
- ✔ The binary representation
  $...b_3 b_2 b_1 b_0 \cdot b_{-1} b_{-2} b_{-3} ...$
  $6.28125 = (110.01001)_2 \Rightarrow 2^2 + 2^1 + 2^{-2} + 2^{-5}$
- ✔ real numbers are stored as a sequence of three bit fields : $(-1)^s \times m \times 2^e$
  s = sign (0,1) m = mantissa (hidden 1.)
  p = exponent (stored p=e+bias=e+127)

$6.28125 = (110.01001)_2 = 1.1001001 \times \underbrace{100}_{2^2}$

$s = 0, \quad p = 2 + 127, \quad m = 100100100...$

Example : 6.28125

Conversion of integral part

div	res	remain
$6/2$	3	0×2^0
$3/2$	1	1×2^1
$1/2$	0	1×2^2

stop when zero on result !

Conversion of decimal part

mult	res	int
0.28125×2	0.5625	0×2^{-1}
0.5625×2	1.1250	1×2^{-2}
0.1250×2	0.2500	0×2^{-3}
0.2500×2	0.5000	0×2^{-4}
0.5000×2	1.0000	1×2^{-5}

stop when zero on decimal part !

binary conversion : limited precision

Example : 11.20

Conversion of integral part

div	res	remain
$11/2$	5	1×2^0
$5/2$	2	1×2^1
$2/2$	1	0×2^3
$1/2$	0	1×2^4

$(11.)_{10} = (1011\cdot)_2$

Conversion of decimal part

mult	res	int
0.20×2	0.40	0×2^{-1}
0.40×2	0.80	0×2^{-2}
0.80×2	1.60	1×2^{-3}
0.60×2	1.20	1×2^{-4}
0.20×2	0.40	0×2^{-5}
$\ldots$	$\ldots$	$\ldots$

$(0.20)_{10} = (\cdot 0011001100110011\cdots)_2$

$+1.01100110011001100110011\cdots 2^3$

Example : 0.42

Conversion of decimal part

mult	res	int
0.42×2	0.84	0×2^{-1}
0.84×2	1.68	1×2^{-2}
0.68×2	1.36	1×2^{-3}
0.36×2	0.72	0×2^{-4}
0.72×2	1.44	1×2^{-5}
0.44×2	0.88	0×2^{-6}
0.88×2	1.76	1×2^{-7}
0.76×2	1.52	1×2^{-8}
0.52×2	1.04	1×2^{-9}
0.04×2	0.08	0×2^{-10}
$\ldots$	$\ldots$	$\ldots$

$(0.42)_{10} = (\cdot 0110101110\cdots)_2$

Shift now the point to the right two times to catch the first $1 \Rightarrow \times 2^{-2}$ $+1.101011\cdots 2^{-2}$

$p = e + 127 = 125$ $m = 101011\cdots$

Number representation : reals (cont.)

- ✔ the first digit of the mantissa is always equal **1**
  many zeros as possible at the end of the binary string
- ✔ one can gain one bit accuracy by avoiding the storage os the mandatory first mantissa **1** bit
  the mantissa has one *hidden bit*
- ✔ the exponent (e) is shifted by an integer bias (127) to avoid negative numbers
  shift avoid us an additional bit to store exponent signal
- ✔ particular cases

  0.0
  by convention, if all bits of the floating point string after the sign are zero, the hidden bit is also zero

  infinity
  exponent bits $\rightarrow$ all 1's
  mantissa bits $\rightarrow$ all 0's

  overflow, underflow
  when exponent takes a value higher than the maximum value or lower than the minimum value that can be described with the available number of bits ($2^8 - 1 = 255$)