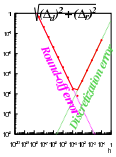
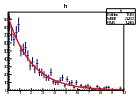
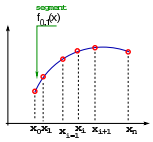
Computational Physics

numerical methods with C++ (and UNIX)

2018-19



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: fernando.barao@tecnico.ulisboa.pt

C++ inheritance example

class C

```
class C: public B {
  public:
    C(float fx=0.0) : B(fx) {
       cout << "Constructor C called" << endl;}
    ~C() {
       cout << ``destructor C called'' << endl;}
    void Print() {
       cout << ``print C (x='' << x << ``)'' << endl;}
   float Norma() {
       return x*x;}
};
```

class Tools

a *fPrint()* method is defined with a base class as argument! Which Print() method will be called?

```
class Tools {
  public:
    static void fPrint(string s) {
      cout << s << endl;}
    static void fPrint(A* p) {
      p->Print();}
 };
```

C++ inheritance example

main program

```
int main() {
  // instantiate objects on heap!
  A* a = new A(3.1415);
  B* b = new B(4.22);
  C* c = new C(1.657);

  // implicit cast to base class
  A* a1 = new C(9878621.09874);

  // cast to B of an existing C object
  B* b1 = (B*)c;

  // managing objects (keeping track)
  int N = 0;
  A** buf = new A*[100];
  buf[N] = a; N++;
  buf[N] = b; N++;
  buf[N] = c; N++;
  buf[N] = a1; N++;

 // Print objects using Tools class
  cout << "\n Tools::fPrint(a);" << endl;
  Tools::fPrint(a);
  cout << "\n Tools::fPrint(buf[1]);" << endl;
  Tools::fPrint(buf[1]);
  cout << "\n Tools::fPrint(c);" << endl;
  Tools::fPrint(c);
  cout << "\n Tools::fPrint(a1);" << endl;
  Tools::fPrint(a1);
  cout << "\n Tools::fPrint(b1);" << endl;
  Tools::fPrint(b1);
```

program output

```
A* a = new A(3.1415);
A::A(float) Constructor A called

B* b = new B(4.22);
A::A(float) Constructor A called
B::B(float) Constructor B called

C* c = new C(1.657);
A::A(float) Constructor A called
B::B(float) Constructor B called
C::C(float) Constructor C called

A* a1 = new C(9878621.09874);
A::A(float) Constructor A called
B::B(float) Constructor B called
C::C(float) Constructor C called

B* b1 = (B*)c;

Tools::fPrint(a);
virtual void A::Print() print A (x=3.1415)

Tools::fPrint(b);
virtual void B::Print() print B (x=4.22)

Tools::fPrint(c);
virtual void C::Print() Print C (x=1.657)

Tools::fPrint(a1);
virtual void C::Print() Print C (x=9.87862e+06)

Tools::fPrint(b1);
virtual void C::Print() Print C (x=1.657)
```


C++ inheritance example

main program

```
(...)

A* d;
Tools::fPrint(d=new C);
buf[N] = d; N++;

cout << "number of pointers stored = " << N << endl;

// delete all heap objects

cout << "\n delete all objects ------------ \n" << endl;
for (int i=0; i<N; i++) {
    cout << "\n delete buf[" <<i << "]" << endl;
    delete buf[i];
}

(...)
```

program output

```
number of pointers stored = 5

delete all objects ------------------

delete buf[0]
virtual A::~A() Destructor A called

delete buf[1]
virtual B::~B() Destructor B called
virtual A::~A() Destructor A called

delete buf[2]
virtual C::~C() Destructor C called
virtual B::~B() Destructor B called
virtual A::~A() Destructor A called

delete buf[3]
virtual C::~C() Destructor C called
virtual B::~B() Destructor B called
virtual A::~A() Destructor A called

delete buf[4]
virtual C::~C() Destructor C called
virtual B::~B() Destructor B called
virtual A::~A() Destructor A called
```

C++ namespaces

- Names in C++ can refer to variables, functions, structures, enumerations, classes and class and structure members. The potential for name conflicts increases when number of code lines become big!

  The usual way to solve this proble in C language was to define some prefix common to all variables belonging to a same package!

- The C++ provides **namespace** facilities to provide greater control over the scope of names.

  The names in one namespace do not conflict with the same names declared in other namespaces

- To access names declared in a given namespace we can use the **scope-resolution operator (::)**

C++ namespaces (cont.)

particle.h (FCOMP namespace)

```
#ifndef __MYH__
#define __MYH__
include <string>
 namespace FCOM {
   //user function
   int addnumbers(int,int);
   int MyFlag=0;
   //user class
   class particle {
     public:
       particle() {}; //default constr
       ...
       void SetMass(double);
     private:
       std::string name;
       double mass;
       int charge;
       ...
   }; //end of class
 } //end of namespace
#endif
```

(particle.C) FCOMP namespace

```
#include "particle.h"
 namespace FCOMP {
   void particle::SetMass(double m) {
     mass = m;
   }
 }
```

user program

```
#include "particle.h"
...
 int main() {
   //set variable value to 101
   FCOMP::MyFlag = 101;
   //call function
   int a = FCOMP::addnumbers(3,10);
   //instantiate object
   FCOMP::particle P;
   P.SetMass(0.511E-3);
 }
```

C++ namespaces (cont.)

- The **using** declaration simplifies the procedure for using the names declared under a given namespace

```
using FCOMP::MyFlag;
```

after this declaration we can use directly the name **MyFlag**.
If we redeclare a variable **MyFlag** now, an error is returned!

- The **using namespace** declaration makes all names defined within the namespace directly accessible!
We do not need anymore the scope-operator to acess them!

```
// all names from FCOMP usable
using namespace FCOMP;
// all names from std usable
using namespace std;
```


Computational Physics

ROOT

A data analysis graphics tool with a C++ interpreter

Fernando Barao, Phys Department IST (Lisbon)

ROOT - outline

- ✔ ROOT installation
- ✔ general concepts
- ✔ interactive use and macros
- ✔ canvas and graphics style
- ✔ histograms and other objects
- ✔ fitting
- ✔ input/ouput
- ✔ using ROOT from user programs

site: `http://root.cern.ch`

Users Guide: `http://root.cern.ch/drupal/content/root-users-guide-600`

ROOT - introduction

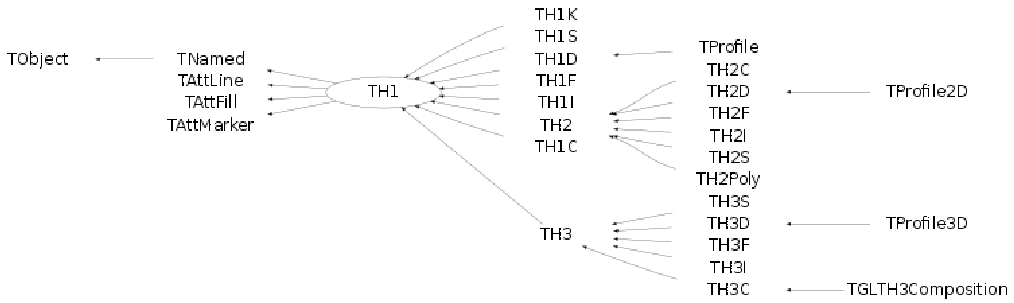
- ✔ ROOT is and object oriented framework designed for solving data handling issues in High Energy Physics such as data storage and data analysis (display, statistics, ...)
- ✔ ROOT was the next step after the PAW data analysis tool developped in Fortran on $90's$ and widely used by physicists
- ✔ ROOT is supported by the CERN organization and it is continuously evolving
- ✔ ROOT is nowadays used in other fields like medecine, finance, astrophysics, ... as a data handling tool

ROOT - categories

many fields/categories covered:

- ✔ base: low level building blocks (TObject,...)
- ✔ container: arrays, lists, trees, maps, ...
- ✔ physics: 2D-vectors, 3D-vectors. Lorentz vector, Lorentz Rotation, N-body phase space generator
- ✔ matrix: general matrices and vectors
- ✔ histograms: 1D,2D and 3D histograms
- ✔ minimization: MINUIT interface,...
- ✔ tree and ntuple: information storage
- ✔ 2D graphics: lines, shapes (rectangles, circles,...), pads, canvases
- ✔ 3D graphics: 3D-polylines, 3D shapes (box, cone,...)
- ✔ detector geometry: monte-carlo simulation and particle tracing
- ✔ graphics user interface (GUI):
- ✔ networking: buttons, menus,...
- ✔ database: MySQL,...
- ✔ documentation

ROOT - TH1 class inheritance



TObject: Mother of all ROOT objects.

The TObject class provides default behaviour and protocol for all objects in the ROOT system. It provides protocol for object I/O, error handling, sorting, inspection, printing, drawing, etc. Every object which inherits from TObject can be stored in the ROOT collection classes.

```
> root -l
 // allocate an array of pointers to TObject
 TObject **OBJ = new TObject*[100]
 // allocate objects
 OBJ[0] = new TH1D()
 OBJ[1] = new TF1()
 OBJ[2] = new TList()
 OBJ[3] = new TMatrixD()
 OBJ[4] = new TCanvas()
 // list objects in memory
 .ls
 // use TBrowser instead of a listing
 new TBrowser()
 // quit
 .q
```

ROOT - start

✔ **root command help**

```
> root --help # get help

Usage: root [-l] [-b] [-n] [-q] [dir] [[file:]data.root] [file1.C ... fileN.C]
Options:
  -b : run in batch mode without graphics
  -n : do not execute logon and logoff macros as specified in .rootrc
  -q : exit after processing command line macro files
  -l : do not show splash screen
  -x : exit on exception
 dir : if dir is a valid directory cd to it before executing

  -?       : print usage
  -h       : print usage
  --help   : print usage
  -config  : print ./configure options
  -memstat : run with memory usage monitoring
```

✔ **start root**

```
> root -l
```

✔ **quit root**

```
> .q
```


ROOT - CINT interpreter

✔ **CINT commands**

```
root> .<command>
            .q : quit
            .? : list of commands
  .x <macro.C> : execute C++ macro
  .L <macro.C> : load macro
 .!<shell cmd> : run shell cmd
                 .!ls  - list files on current directory
                 .!pwd - print current directory name
         .func : list all functions
```

✔ **ROOT global pointers**

gROOT instance of the *TROOT class* works as an entry point to the ROOT system, providing access to the stored ROOT objects

gSystem defines an interface to the underlying operating system (*TUnixSystem*)

gStyle defines attributes of objects: lines, canvas, pad, histograms,...

gRandom instance of *TRandom3 class* providing a quick access to random number generator

ROOT - calculator

□ **ROOT used as a calculator**

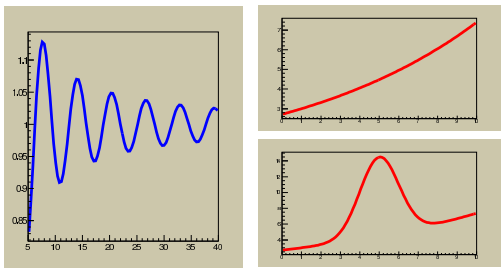
```
> root -l
root [0] 7+2/6 //do not put ";" at the end to get answer
(const int)7
root [1] 7+2/6.
(const double)7.3333333333333304e+00
root [2] 1>2 //evaluate expression
(const int)0
root [3] TMath::Pi()
(Double_t)3.14159265358979312e+00
root [4] TMath::Sin(10.*TMath::Pi()/180.) //compute sin(10 degrees)
(Double_t)1.73648177666930331e-01

root [18] double result = 0.
(const double)0.00000000000000000e+00
root [19] for (int i=0;i<10;i++) {result += TMath::Power(0.5,i);}
root [20] result
(double)1.99804687500000000e+00
```


ROOT - graphics window

□ **The graphics window in ROOT is made using the *TCanvas class***
□ **Let's open a canvas and divide it in three *pads* where the graphics objects will be drawn**

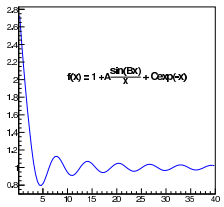
```
[0] gROOT->Reset();
[1] gStyle->SetOptTitle(0);
[2] TF1 *f1 = new TF1("f1","1.+ [0]*sin([1]*x)/x + [2]*exp(-x)",0.1, 40.);
[3] f1->SetParameters(1.,1.,1.);
[4] f1->SetLineColor(kBlue);
[5] f1->SetRange(5.,40.);
[6] TCanvas *c = new TCanvas("c", "Phys Comput canvas", 0, 0, 900, 500);
[7] TPad *pad1 = new TPad("pad1","The 2nd pad",0.02,0.02,0.48,0.98,21);
[8] TPad *pad2 = new TPad("pad2","The 2nd pad",0.51,0.52,0.98,0.98,21);
[9] TPad *pad3 = new TPad("pad3","The 3rd pad",0.51,0.02,0.98,0.49,21);
[10] pad1->Draw(); pad2->Draw(); pad3->Draw();
[11] pad1->cd(); f1->SetLineWidth(4); f1->DrawCopy();
[12] TF1 *f2 = new TF1("f2","expo(0)",0.,10.); //expo=exp(A+Bx)
[13] f2->SetParameters(1., 0.1);
[14] pad2->cd(); f2->SetLineWidth(4); f2->Draw();
[15] TF1 *f3 = new TF1("f3","expo(0)+gaus(2)",0.,10.);
[16] f3->SetParameters(1., 0.1, 10., 5., 1.); //exp+gau
[17] pad3->cd(); f3->SetLineWidth(4); f3->Draw();
[18] c->Modified();
```

root.cern.ch/root/html/TFormula.html



- **ROOT can be used to plot functions: classes TF1, TF2**
- **Plot function:** $f_1(x) = A\frac{sin(Bx)}{x} + Ce^x$

```
[0] gROOT->Reset();
[1] gStyle->SetOptTitle(0);
[2] TCanvas *c = new TCanvas("c", "Phys Comput canvas", 0, 0, 500, 500);
[3] TF1 *f1 = new TF1("f1","1.+ [0]*sin([1]*x)/x + [2]*exp(-x)",0.1, 40.);
[4] f1->SetParameters(1.,1.,1.);
[5] f1->SetLineColor(2);
[6] f1->GetHistogram()->GetXaxis()->SetTitle("x");
[7] f1->Draw();
[8] TLatex l(10.,2.0,''f(x) = 1 +A#frac{sin(Bx)}{x} + Cexp(-x)'');
[9] l.SetTextSize(0.04);
[10] l.Draw();
[11] c->Modified();
[12] c->SaveAs("Sfunctionplotter.eps");
```



- **Plot function:** $f_2(x,y) = \frac{sin(x)\cdot sin(y)}{x\cdot y}$

```
[13] TF2 *f2 = new TF2("f2","sin(x)*sin(y)/(x*y)",0,5,0,5);
[14] gPad->SetTheta(25);
[15] gPad->SetPhi(-110);
[16] gPad->SetLogz();
[17] f2->Draw("surf1"); //"", plot contours
```