

# Root 2008

[www.if.usp.br/suaide](http://www.if.usp.br/suaide)

Alexandre Suaide

Aula 4

# Programa

- **Aula 1**
  - Introdução ao c++ e ROOT
  - Conceito de classe e objeto
  - Básico de gráficos e funções no ROOT
- **Aula 2**
  - Mais gráficos e funções
  - Histogramas de 1 e 2D
  - Ajustes de funções, legendas, etc.
  - Escrevendo programas simples: Monte Carlo e simulações
- **Aula 3**
  - Referências e ponteiros
  - Nomes e memória
  - Programação mais complexa: mais Monte Carlo
- **Aula 4**
  - I/O no ROOT
  - Mais programação no ROOT
  - Compilando com o ROOT

## Gravando objetos em arquivos .root

- Ok, criei meus gráficos, histogramas, funções ou qualquer outro objeto do ROOT. Como eu posso armazená-los?
- O Root possui um sistema de arquivos complexos, onde TODOS os seus objetos podem ser salvos em um arquivo com estrutura de diretórios.
- TFile
  - Classe do ROOT para abrir, ler, gravar e manipular arquivos .root

# TFile: Gravando objetos em um arquivo

- **Uso**

- `TFile f("nome_do_arquivo","opções");`
- `TFile* f = new TFile("nome_do_arquivo","opções");`

- **Opções**

- **NEW** ou **CREATE** - Abre o arquivo para escrita. Se o arquivo já existir, nada acontece
- **RECREATE** - Abre o arquivo para escrita. Se o arquivo já existir, escreve por cima.
- **READ** (ou sem opção) - Abre o arquivo para leitura.

- **Gravando e lendo objetos**

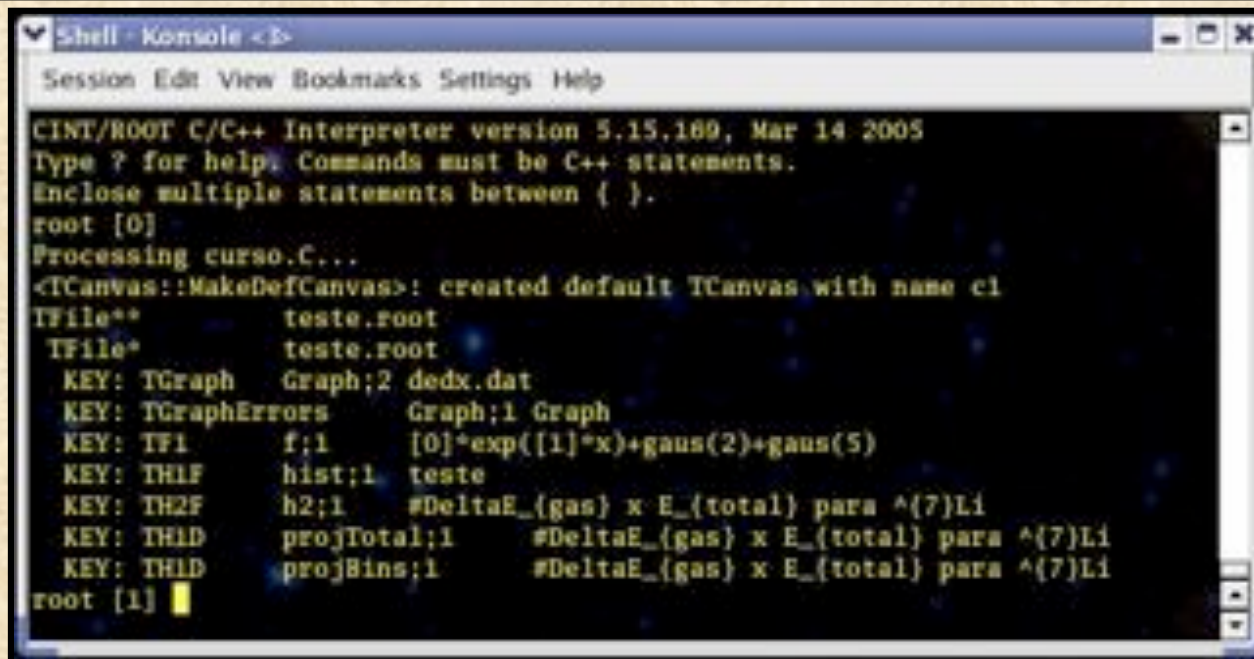
- Para gravar, use a função `Write()` dos objetos a serem gravados
- Para ler, use a função `Get("nome")` do objeto `TFile`



# Gravando objetos no arquivo (Write)

- Criamos vários objetos em vários exemplos. Como podemos gravá-los em um arquivo?

```
TFile file("teste.root","RECREATE");  
file.cd(); // muda o diretorio padrão para o arquivo criado  
g->Write();  
dedx->Write();  
f->Write();  
h->Write();  
h2->Write();  
projTotal->Write();  
projBins->Write();  
file.ls(); // lista o conteudo do arquivo  
file.Close(); // fecha o arquivo
```



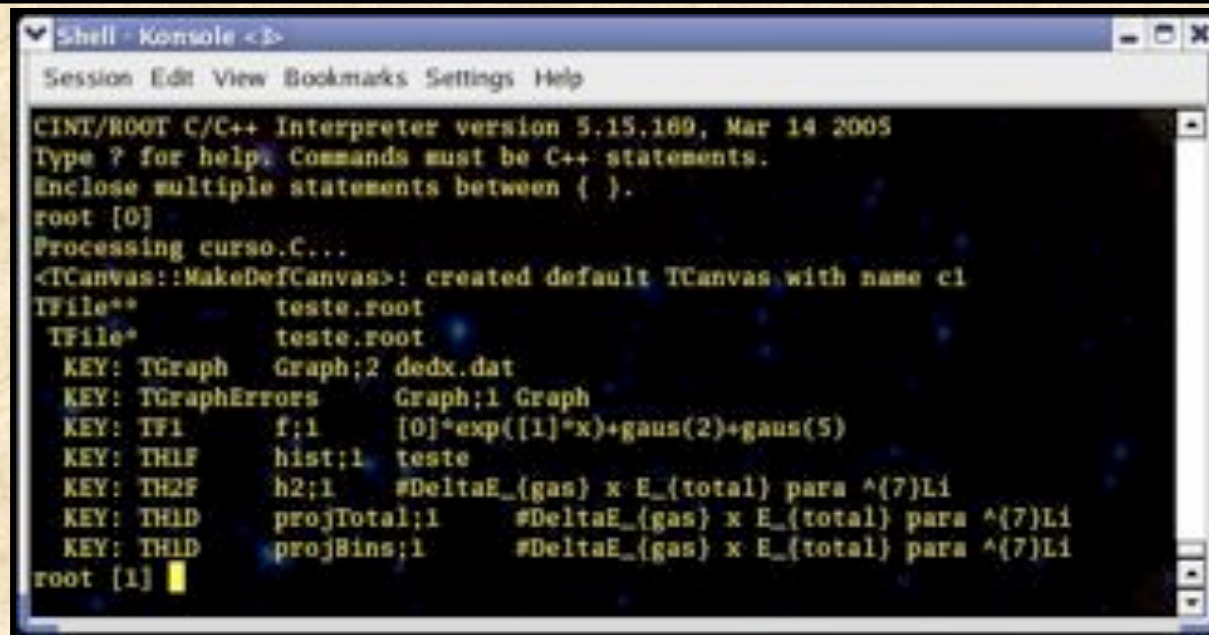
The screenshot shows a terminal window titled "Shell - Konsole <>". The window contains the following text:

```
Session Edit View Bookmarks Settings Help  
CINT/ROOT C/C++ Interpreter version 5.15.189, Mar 14 2005  
Type ? for help. Commands must be C++ statements.  
Enclose multiple statements between { }.  
root [0]  
Processing curso.C...  
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1  
TFile**      teste.root  
TFile*       teste.root  
KEY: TGraph  Graph;2 dedx.dat  
KEY: TGraphErrors  Graph;1 Graph  
KEY: TF1      f;1      [0]^exp([1]^x)+gaus(2)+gaus(5)  
KEY: TH1F     hist;1  teste  
KEY: TH2F     h2;1    #DeltaE_{gas} x E_{total} para ^{7}Li  
KEY: TH1D     projTotal;1  #DeltaE_{gas} x E_{total} para ^{7}Li  
KEY: TH1D     projBins;1   #DeltaE_{gas} x E_{total} para ^{7}Li  
root [1]
```

# Lendo objetos do arquivo (Get)

- Como eu recupero o histograma "h2" do arquivo?
  - O importante são os nomes dos objetos, que se obtém a partir do método `ls()` da classe `TFile`.

```
TFile file("teste.root"); // abrir para leitura
file.cd();
TH2F* myhist = (TH2F*)file.Get("h2");
myhist->Draw("colz");
```
- **Importante!!!**
  - Não feche o arquivo enquanto estiver trabalhando com o histograma, pois o mesmo será removido da memória



```
Shell - Konsole <D>
Session Edit View Bookmarks Settings Help
CINT/ROOT C/C++ Interpreter version 5.15.189, Mar 14 2005
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
Processing curso.C...
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
TFile**      teste.root
TFile*       teste.root
KEY: TGraph   Graph;2 dedx.dat
KEY: TGraphErrors  Graph;1 Graph
KEY: TF1      f;1      [0]*exp([1]*x)+gaus(2)+gaus(5)
KEY: TH1F     hist;1  teste
KEY: TH2F     h2;1    #DeltaE_{gas} x E_{total} para ^{7}Li
KEY: TH1D     projTotal;1  #DeltaE_{gas} x E_{total} para ^{7}Li
KEY: TH1D     projBins;1   #DeltaE_{gas} x E_{total} para ^{7}Li
root [1]
```



## Expandindo as possibilidades: criando novas classes no ROOT

- Muitas vezes fazemos uma série de procedimentos repetidos que seriam interessantes que fizessem parte do próprio objeto
  - Por exemplo, cálculo de resíduos

- Podemos criar novas classes no ROOT que incorporem novas funcionalidades e, a partir delas, trabalhar de forma mais eficiente
  - Exemplo:
    - MG (meu gráfico)

# Classe MG (Meu Gráfico)

- **Requisitos**

- Deve conter todas as características de um gráfico com incertezas
  - Deve derivar de TGraphErrors
- Devemos ser capazes de criar objetos de dois modos
  - Ler dados de um arquivo
  - Fornecer os vetores com os dados
- Deve conter os seguintes métodos
  - Residuos → calcula gráfico de resíduos em relação a uma função
  - curvaChi2 → fornece curva de Chi2red em relação a uma função, variando um dos parâmetros



# Definindo a classe (arquivo MG.C)

Nome da classe

Derivada publica de TGraphErrors

Dois contrutores

Valor default para o parâmetro (pode ser excluido da chamada)

```
class MG: public TGraphErrors
{
    public:
        MG(char*, char* = "%lg %lg");
        MG(int, double*, double*, double * = 0, double* = 0);
        virtual ~MG();

        MG*      residuos(TF1*, bool = false);
        MG*      curvaChi2(TF1*, int, float, float, bool = false);
};
```

Definição termina com ;

Duas funções definidas

Destrutor (deve estar presente sempre e ser virtual)

# Definindo as funções (arquivo MG.C)

## Construtores e destrutores

```
MG::MG(char* file, char* formato):TGraphErrors(file, formato)
{
    SetMarkerStyle(20);
    SetLineColor(2);
    SetMarkerColor(2);
}
MG::MG(int n, double* x, double *y, double *ex,
        double *ey):TGraphErrors(n, x, y, ex, ey)
{
    SetMarkerStyle(20);
    SetLineColor(2);
    SetMarkerColor(2);
}
MG::~MG()
{
}
```

Esta sintaxe diz que, quando o construtor da classe for chamado, também chamamos o construtor da classe TGraphErrors

# Definindo as funções (arquivo MG.C)

## Função residuos

```
MG* MG::residuos(TF1* func, bool desenha)
{
    double* x = GetX();
    double* y = GetY();
    double* ex = GetEX();
    double* ey = GetEY();
    int N = GetN();
    double YM[1000],EY[1000];
    for(int i = 0;i<N;i++) {
        double teoria = func->Eval(x[i]);
        YM[i] = (y[i]-teoria)/ey[i];
        EY[i] = 1;
    }
    MG* r = new MG(N,x,YM,ex,EY);
    if(desenha) {
        new TCanvas();
        r->Draw("AP");
    }
    return r;
}
```

Note que podemos usar qualquer função já presente da classe TGraphErrors pois esta é derivada daquela.

Não precisamos nos preocupar com ponteiros, pois o c++ entende que a função é deste objeto



# Definindo as funções (arquivo MG.C)

## Função curvaChi2

```
MG* MG::curvaChi2(TF1* func, int par, float pmin, float pmax, bool desenha)
{
    float dp = (pmax-pmin)/100.0;
    float NP = (float)GetN();
    double x[100], y[100];
    int i = 0;
    for(float p = pmin; p<=pmax; p+=dp) {
        func->SetParameter(par,p);
        float chisq = Chisquare(func)/(NP-1);
        x[i] = p;
        y[i] = chisq;
        i++;
    }
    MG *r = new MG(100,x,y,0,0);
    if(desenha) {
        new TCanvas();
        r->Draw("AP");
    }
    return r;
}
```

# Usando esta classe

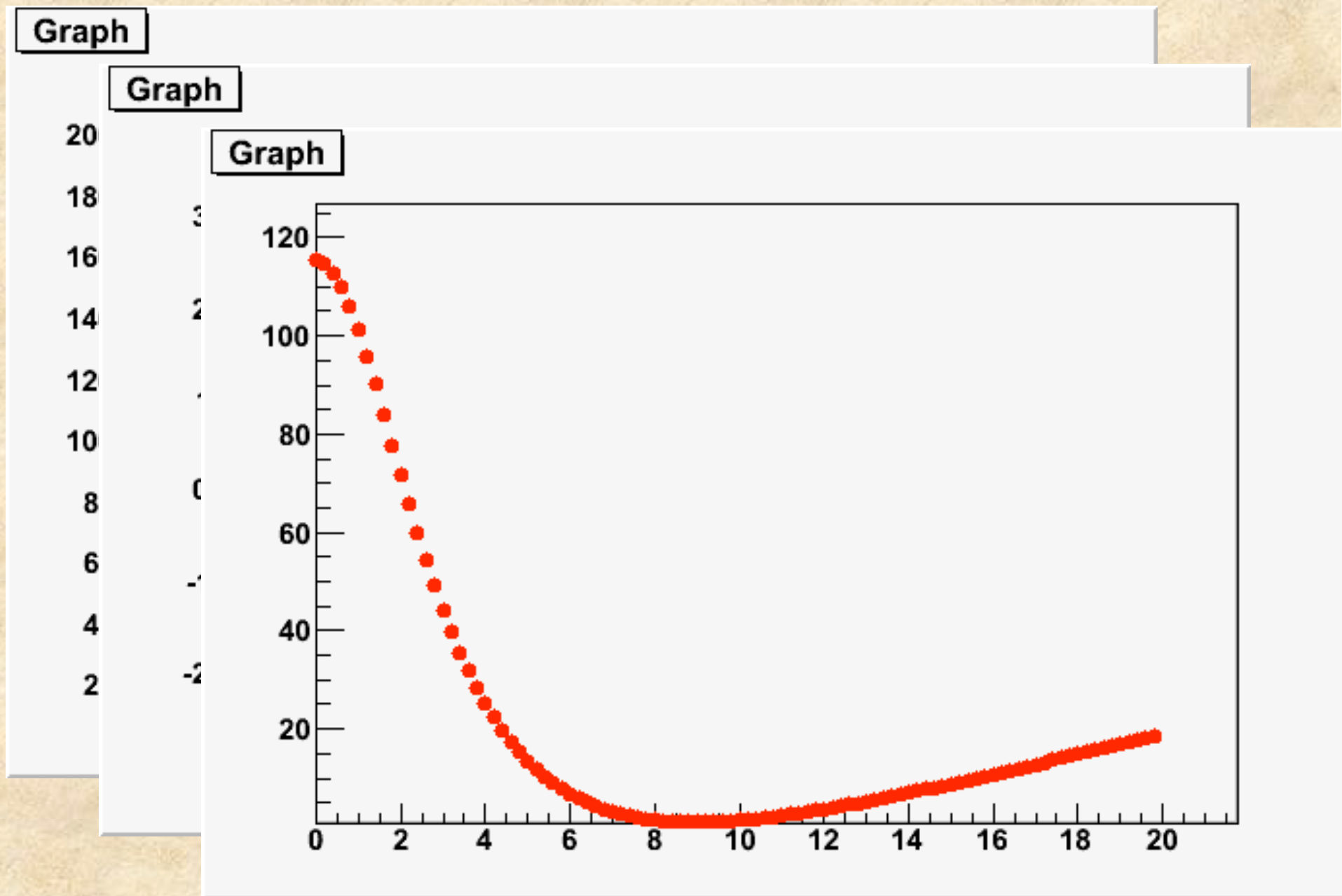
- No prompt do ROOT digite:

```
Root [0] .L MG.C
```

```
Root [1] .x exemplo_MG.C
```

```
exemplo_MG()  
{  
  MG* g = new MG("fit.dat", "%lg %lg %lg");  
  g->Draw("a p");  
  TF1* f = new TF1("f", "[0]/(x*[1]*sqrt([2]^2+(x*[3]-1/(x*[1]))^2))", 3000, 7000);  
  f->SetParameters(5, 1e-6, 5, 30e-3);  
  g->Fit(f);  
  
  MG* r = g->residuos(f, true);  
  
  MG* c1 = g->curvaChi2(f, 2, 0, 20, true);  
}
```

# Três gráficos surgirão





## Ou seja...

- O ROOT, por ser uma interface com linguagem c++ permite todas as artimanhas de programação desta linguagem
- Com o tempo vamos desenvolvendo bibliotecas de programas, funções e classes derivadas (ou novas) que permitem personalizar o trabalho com o ROOT

# Uma última classe interessante do ROOT

## TControlBar

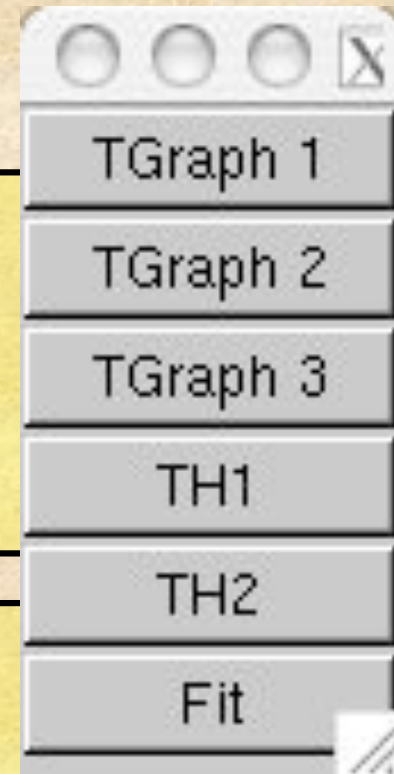
- O ROOT possui várias classes para fazer interfaces gráficas com o usuário
  - Permite fazer programação gráfica completa, abrir janelas, criar diálogos, etc.
- TControlBar
    - Classe para fazer uma barra de botões personalizados
    - Bem simples, mas eficiente em muitos casos

# exemplo\_control.C

- No prompt do ROOT digite:

```
Root [0] .x exemplo_control.C
```

```
TControlBar* bar;  
exemplo_control()  
{  
    bar = new TControlBar("vertical", "meus exemplos");  
    bar->AddButton("TGraph 1", ".x exemplo_TGraph_1.C", "Exemplo TGraph 1");  
    bar->AddButton("TGraph 2", ".x exemplo_TGraph_2.C", "Exemplo TGraph 2");  
    bar->AddButton("TGraph 3", ".x exemplo_TGraph_3.C", "Exemplo TGraph 3");  
    bar->AddButton("TH1", ".x exemplo_TH1.C", "Exemplo Hist 1D");  
    bar->AddButton("TH2", ".x exemplo_TH2.C", "Exemplo Hist 2D");  
    bar->AddButton("Fit", ".x exemplo_fit_2.C", "Exemplo Fit");  
    bar->Show();  
}
```





# Compilando programas no ROOT

- **Porque compilar?**
  - No ROOT, quando executamos um programa (.x, etc) cada linha do programa é compilada quando executada, ou seja, a execução é sempre em 2 passos
    - Queda de performance, principalmente se o programa for complexo
  - O Root permite compilação de forma muito simples
    - Contudo, precisamos de um compilador c++ externo
      - Linux, Mac OS → gcc
      - Windows → VC++

# Como compilar?

- Com o ROOT
  - Coloque ++ após o comando (.L) de carregar um programa na memória

```
Root [0] .L MG.C++
```

•É importante notar, contudo, que agora estamos chamando um compilador externo, que tem sintaxe de c++ mais rigorosa que o ROOT

•Precisamos seguir a sintaxe a risca, inclusive colocar os includes do ROOT para as suas classes, pois o gcc não sabe que elas existem.

•ROOT chamará o gcc (ou VC) e compilará o programa.

Se tudo der certo, vai haver um arquivo MG\_C.so (ou dll) no disco

# Modificando o arquivo MG.C

```
#include "TGraphErrors.h"  
#include "TF1.h"  
#include "TCanvas.h"
```

```
class MG: public TGraphErrors  
{  
    public:  
        MG(char*, char* = "%lg %lg");  
        MG(int, double*, double*, double * = 0, double* = 0);  
    virtual ~MG();  
  
    MG*      residuos(TF1*, bool = false);  
    MG*      curvaChi2(TF1*, int, float, float, bool = false);  
};
```

•Precisamos dizer ao compilador onde as classes do ROOT que são utilizadas neste programa estão definidas

•Como eu sei quais includes colocar?



# Descobrimos os #includes

- Olhar página de documentação do ROOT
  - Exemplo de TGraphErrors



The screenshot shows a web browser window displaying the ROOT documentation for the TGraphErrors class. The browser's address bar shows the URL <http://www.itp.uni-bonn.de/~suaide/>. The page content includes a navigation menu with links for 'Location', 'Quick Links', 'Source', and 'Sections'. The main heading is 'class TGraphErrors: public TGraph'. A sidebar on the right shows the class hierarchy, including the library 'libGraf' and the include file '#include "TGraphErrors.h"'. A red arrow points to this sidebar. Below the heading, there is a description of the class and a 'Picture' button.

Location: [ROOT](#) » [GRAF2D](#) » [GRAF](#) » [TGraphErrors](#)

Quick Links: [ROOT Homepage](#) | [Class Index](#) | [Class Hierarchy](#) |  [Search](#)

Source: [header file](#) | [source file](#) | [viewCVS header](#) | [viewCVS source](#)

Sections: [class description](#) | [function members](#) | [data members](#) | [class charts](#)

class TGraphErrors

library: libGraf  
#include "TGraphErrors.h"

Display options:  
 Show inherited  
 Show non-public  
[\[↑ Top\]](#) [\[? Help\]](#)

**class TGraphErrors: public TGraph**

TGraphErrors class

A TGraphErrors is a TGraph with error bars. The various format options to draw a TGraphErrors are explained in [TGraphErrors::Paint](#).  
The picture below gives an example:

[Picture](#) [Source](#)

## Carregando um arquivo compilado no ROOT

- Use o objeto `gSystem`, da classe `Tsystem`
- No Root, digite, ao invés de carregar o arquivo `.C`:

```
Root [0] gSystem->Load("MC_C.so");
```

## O que foi visto até aqui

- Usar o ROOT se resume em:
  - Saber um pouco de c++
  - Conhecer as classes do ROOT
    - Muito pode ser feito com algumas classes básicas
      - Histogramas, gráficos e funções
  - Ser criativo para aproveitar as ferramentas
- Como eu posso avançar?
  - Tentando...
    - Nesse ambiente, tentar resolver os problemas é a melhor forma de aprendizado...
  - e perguntando
    - Sintam-se à vontade para me encher a paciência ☺