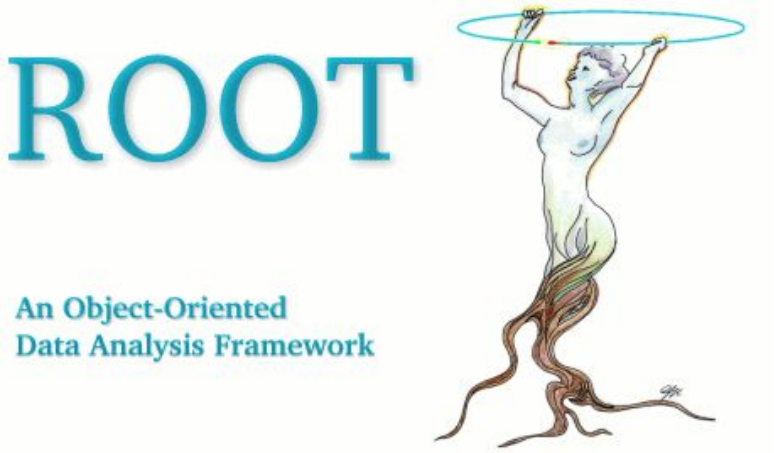
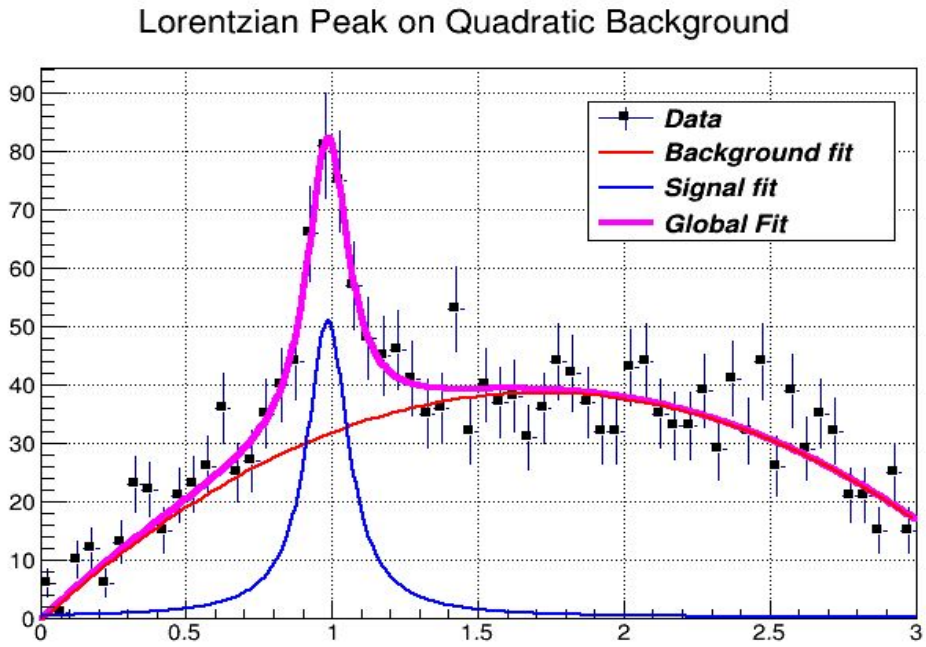
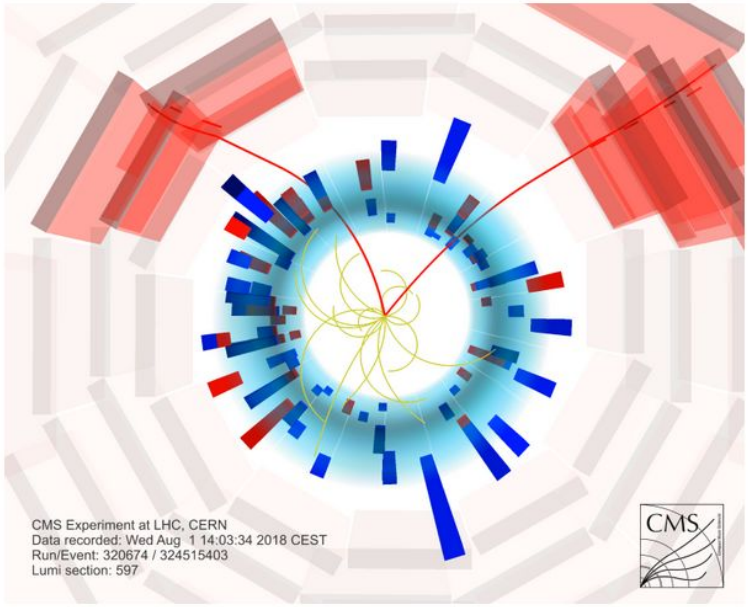






Big data With ROOT CERN

Jhovanny Andres Mejia Guisao

Centro Interdisciplinario de Investigación
y Enseñanza de la Ciencia (CIIEC)

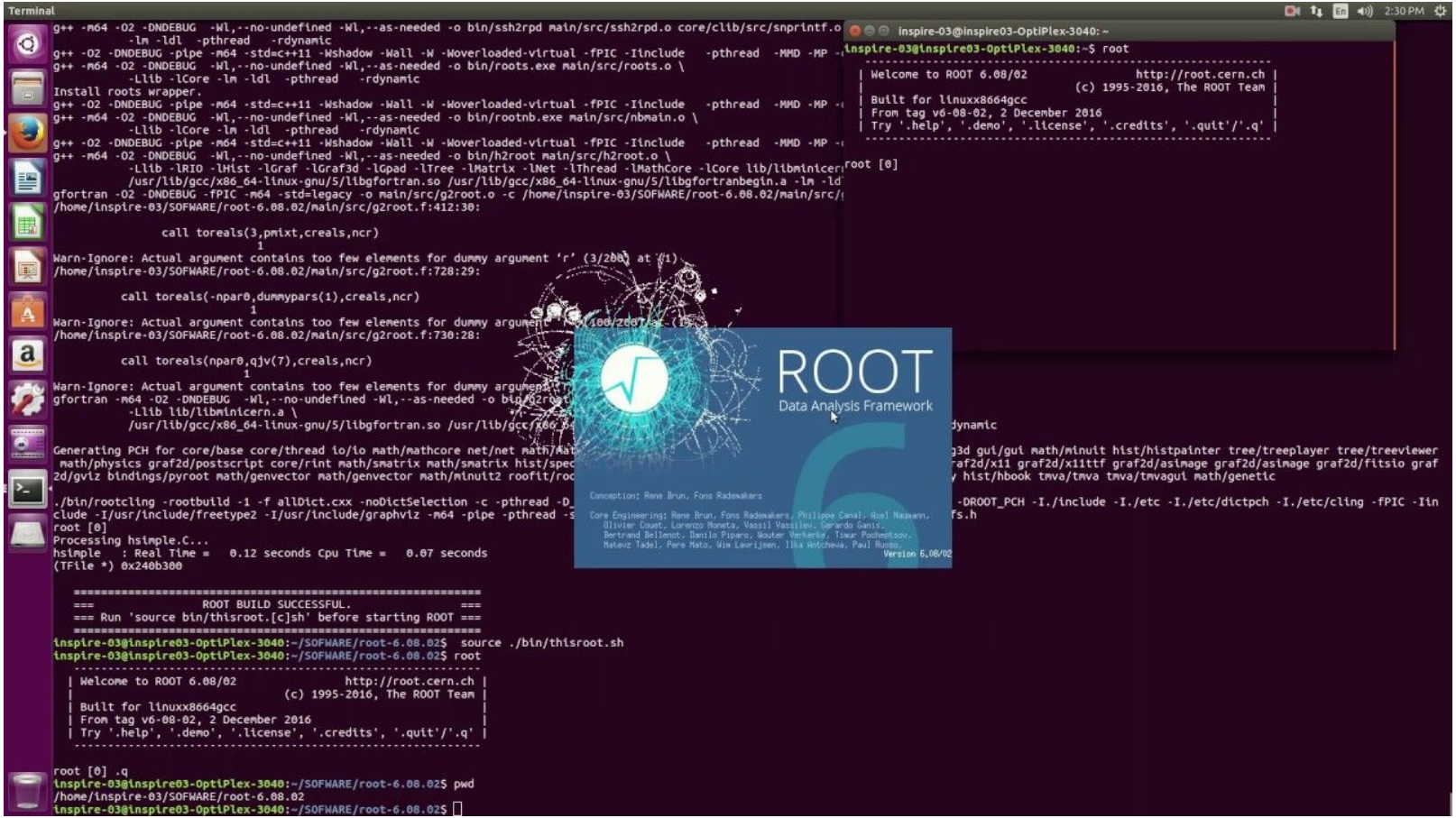
Motivation

What we hope to discuss about scientific data analysis?

- Advanced graphical user interface
- Interpreter for the C++ programming language
- Persistency mechanism for C++ objects
- Used to write every year petabytes of data recorded by the Large Hadron Collider experiments

Input and plotting of data from measurements and fitting of analytical functions.

ROOT Basics


Terminal
inspire-03@inspire03-OptiPlex-3040: ~
inspire-03@inspire03-OptiPlex-3040:~$ root
| Welcome to ROOT 6.08/02 http://root.cern.ch |
| (c) 1995-2016, The ROOT Team |
| Built for linuxx8664gcc |
| From tag v6-08-02, 2 December 2016 |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
root [0]
ROOT
Data Analysis Framework
Conception: Rene Brun, Fons Rademakers
Version 6.08/02
root [0] .q
inspire-03@inspire03-OptiPlex-3040:~/SOFWARE/root-6.08.02$ pwd
/home/inspire-03/SOFWARE/root-6.08.02
inspire-03@inspire03-OptiPlex-3040:~/SOFWARE/root-6.08.02$

ROOT Basics: The ROOT Prompt

- C++ is a compiled language
  - A compiler is used to translate source code into machine instructions
- ROOT provides a C++ **interpreter**
  - Interactive C++, without the need of a compiler, like Python, Ruby, Haskell …
    - Code is **Just-in-Time compiled!**
  - Allows reflection (inspect at runtime layout of classes)
  - Is started with the command:

```
root
```

  - **The interactive shell is also called “ROOT prompt” or “ROOT interactive prompt”**

ROOT As a Calculator

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \ldots$$

$$= \sum_{n=0}^{\infty} x^n$$

Here we make a step forward.
We declare **variables** and use a ***for*** control structure.

```
root [0] double x=.5
(double) 0.5
root [1] int N=30
(int) 30
root [2] double gs=0;
```

```
root [3] for (int i=0;i<N;++i) gs += pow(x,i)
root [4] std::abs(gs - (1/(1-x)))
(Double_t) 1.86265e-09
```

Controlling ROOT

- Special commands which are not C++ can be typed at the prompt, they start with a “.”

```
root [1] .<command>
```

- For example:
  - To quit root use **.q**
  - To issue a shell command use **.! <OS_command>**
  - To load a macro use **.L <file_name>** (see following slides about macros)
  - **.help** or **.?** gives the full list

Ex Tempore Exercise

- Fire up ROOT
- Verify it works as a calculator
- Inspect the help
- Quit

Exercise

- ROOT provides mathematical functions, for example the widely known and adopted Gaussian

- For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine


```
root [0] double x=0
root [1] exp(-x*x*.5) - TMath::Gaus(x)
[...]
```

Solution

- For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine

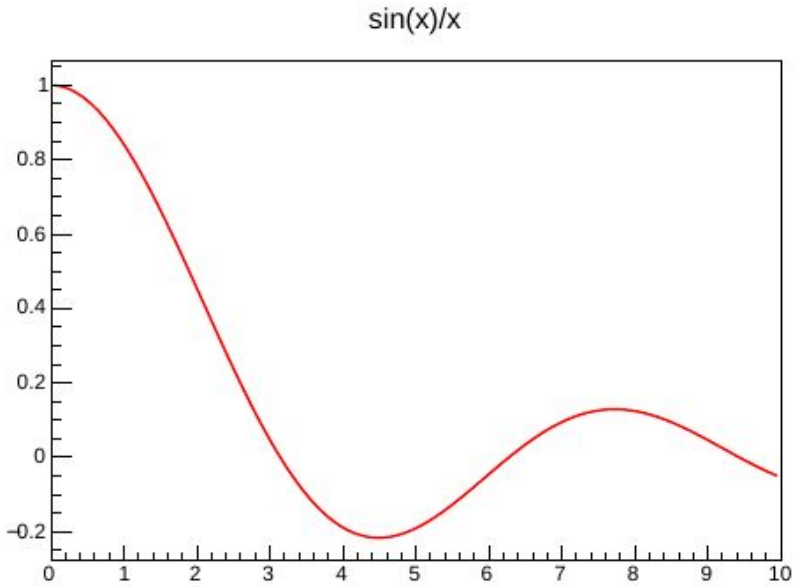
```
root [0] double x=0
root [1] exp(-x*x*.5) - TMath::Gaus(x)
[…]
```

- Many possible ways of solving this! E.g:

```
root [0] for (auto v : {0.,1.,10.,20.}) cout << v << “ ” <<
exp(-v*v*.5) - TMath::Gaus(v) << endl
```

my first function

```
root [0] auto fa1 = new TF1("fa1","sin(x)/x",0,10);
root [1] fa1->Draw();
[...]
```



Please, take a look at the "toolbar"

1) save as png,pdf,root,.....
2) change the line color, style
3) change the tf1 range
4) change the tf1 function (cos(x)+x)?
4) change the x-label name
6) continue to explore the options

my Error function

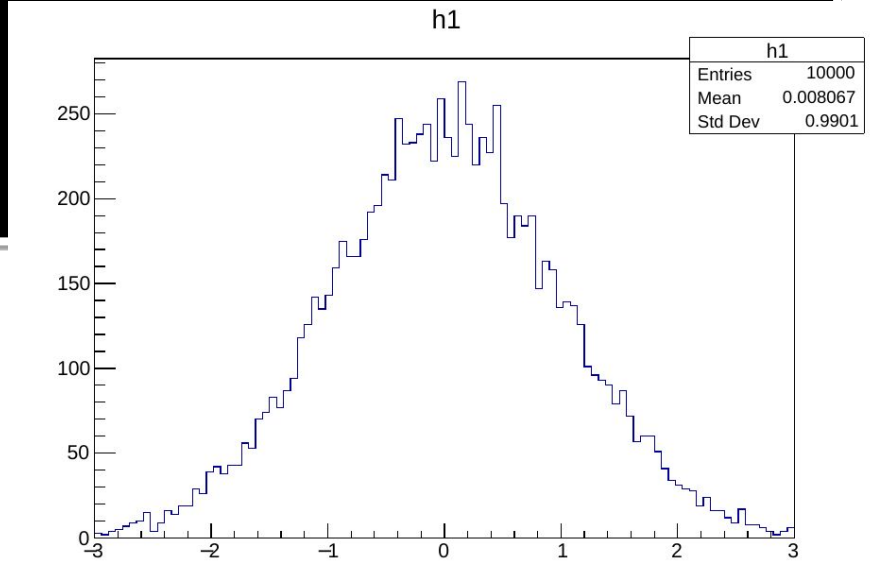
```
root [0] auto fa2 = new TF1("fa2","TMath::Erf(x)",0,2);
root [1] fa2->Draw();
[...]
```

Trigger Effy, https://en.wikipedia.org/wiki/Error_function,

```
root [13] auto fa2 = new TF1("fa2","TMath::Erf(x)",0,2);
root [14] fa2->Draw();
auto fa2 = new TF1("fa2","TMath::Erf(-x)",0,2);
root [15] fa2->Draw();
root [16] auto fa2 = new TF1("fa2","(TMath::Erf((-x+1.0)))",0,2);
root [17] fa2->Draw();
root [18] auto fa2 = new
TF1("fa2","(TMath::Erf((-x+1.0)/0.035))",0,2);
root [19] fa2->Draw();
```

my first histogram

```
root [0] TH1F h1("h1","h1",100,-3,3);
root [1] h1.FillRandom("gaus",10000);
root [2] h1->Draw();
ROOT_prompt_2:1:3: error: member reference type 'TH1F' is not a
pointer; did you mean to use '.'?
h1->Draw();
~~^~
  .
root [3] h1.Draw();
```



Please, take a look at the "toolbar", again.

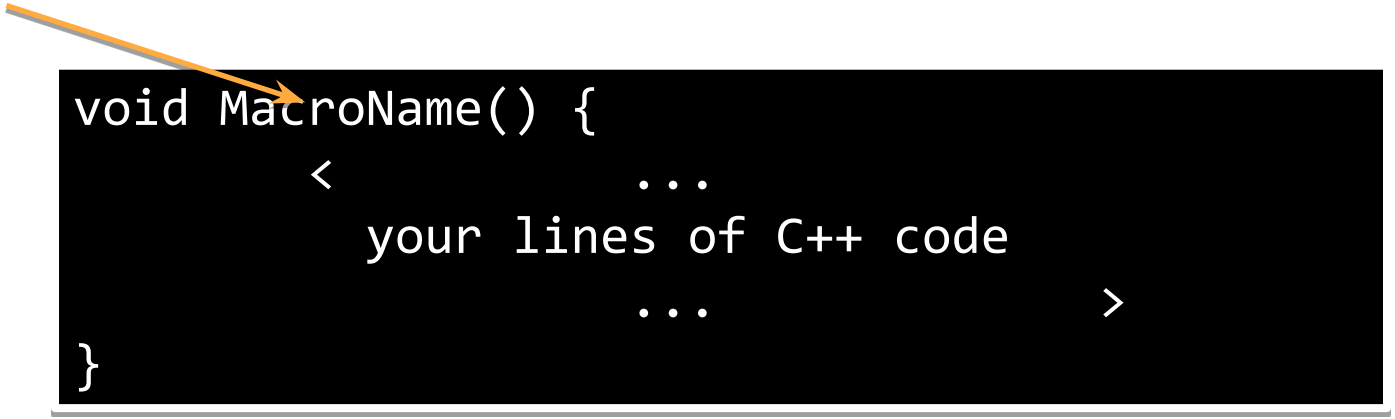
User function

[a]*exp(-0.5*((x-[b])/[c])*((x-[b])/[c]))

ROOT Macros

- We have seen how to interactively type lines at the prompt
- The next step is to write “ROOT Macros” – lightweight programs
- The general structure for a macro stored in file *MacroName.C* is:

Function, no main, same name as the file



```
void MacroName() {
        <           ...
          your lines of C++ code
                    ...             >
}
```

Unnamed ROOT Macros

- Macros can also be defined with no name
- Cannot be called as functions!
  - See next slide :)

```
{
        <           ...
          your lines of C++ code
                    ...             >
}
```

Running a Macro

- A macro is executed at the system prompt by typing:

```
> root MacroName.C
```

- or executed at the ROOT prompt using .x:

```
> root
root [0] .x MacroName.C
```

- or it can be loaded into a ROOT session and then be run by typing:

```
root [0] .L MacroName.C
root [1] MacroName();
```

Time for Exercises

- Go back to the geometric series example we executed at the prompt
- Make a macro out of it and run it

Examples:
geometrica_macro.C
geometrica_macroName.C
¿Como volver esto un codigo puramente cpp?
¿geometrica_macroMain.C?

Interpretation and Compilation

- We have seen how ROOT interprets and “just in time compiles” code. ROOT also allows to compile code “traditionally”. At the ROOT prompt:

```
root [1] .L macro1.C+
root [2] macro1()
```

Generate shared library and execute function

- ROOT libraries can also be used to produce standalone, compiled applications:

```
int main() {
  ExampleMacro();
  return 0;
}
```

```
> g++ -o ExampleMacro ExampleMacro.C `root-config --cflags --libs`
> ./ExampleMacro
```

Un paréntesis útil: Argumentos de línea de comandos

Por suerte, la interfaz para **transmitir argumentos a una función main() se ha estandarizado en C++**, así que la emisión y recepción de argumentos puede hacerse de manera casi mecánica.

Los argumentos transmitidos a main(), como todos los argumentos de función, **deben declararse como parte de la definición de la función**.

int main(int argc, char *argv[])

Sin importar cuántos argumentos se mecanografíen en la línea de comandos, main() sólo necesita las dos piezas de información estándares proporcionadas por **argc y argv**; el número de elementos en la línea de comandos y la lista de direcciones iniciales que indican dónde se almacena en la actualidad cada argumento.
argc (abreviatura para contador de argumento)
argv (abreviatura para valores de argumentos)

Example: Parentesis_ARG

Cualquier argumento mecanografiado en una línea de comandos se considera una cadena en C. Si se desea transmitir datos numéricos a main(), depende de usted convertir la cadena transmitida en su contraparte numérica

Macro myGaussfun

- Make a macro out of it and run it

Examples:
myGaussfun.C
¿Como volver esto un codigo puramente cpp?

My first Canvas

```
root[0] new TCanvas                    ← "quick" creation of graphical window with generic properties
(class TCanvas*)0x2c1e5e0              ← Notice: automatic name assignment ("c1")
root[1] c1->Set
```

```
root[1] c1->SetTitle ("HelloCanvas")
root[2] c1->GetTitle ()
(const char* 0x1557339)"HelloCanvas"
root[3] c1->ls ()
root[4] c1->Close()
root[5] TCanvas c2                     ← We create a new window. Before we used pointer. Now – object.
root[6] c2.GetName()
(const char* 0x16632b1)"c1_n2"         ← Title is different than variable name!
root[7] TCanvas c3 (
```

Multitude of constructors

```
TCanvas TCanvas(Bool_t build = kTRUE)
TCanvas TCanvas(const char* name, const char* title = "", Int_t form = 1)
TCanvas TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
TCanvas TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy,
Int_t ww, Int_t wh)
TCanvas TCanvas(const char* name, Int_t ww, Int_t wh, Int_t winid)
root[7] TCanvas c3 ("c3canvas", "My canvas", 600, 400);
```

Proper name of object (within C++).

"Name" (identifier) (within ROOT).

Displayed title (just a c-string)

```
root[8] c3Canvas
(class TCanvas*)0x16964d0
root[9] TCanvas* c4 = new TCanvas ("c4canv", "2nd Canvas", 600, 400);
```

Name as identifier or replacement of C++ name

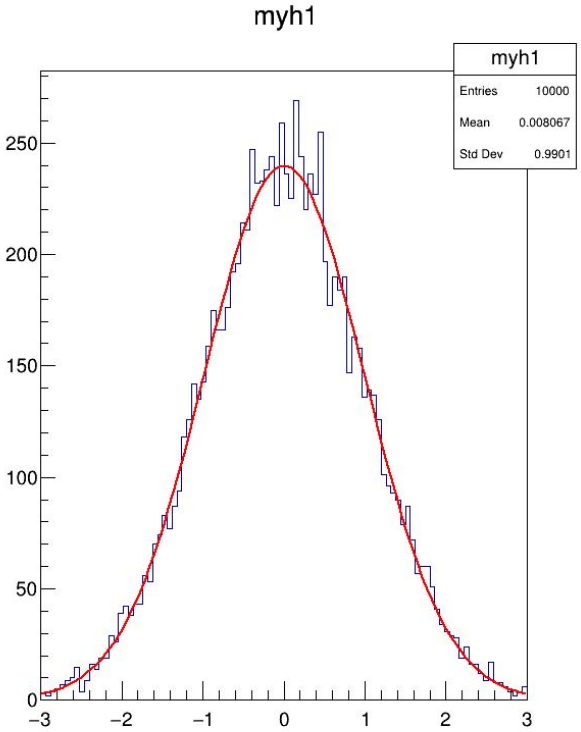
Dynamic allocation (we then use a pointer to an object)

[TCanvas]

My first Canvas

https://root.cern.ch/doc/master/classTCanvas.html

```
auto myc1 = new
TCanvas("myc1","myc1",600,800);
myc1->cd();
myc1->SetLeftMargin(0.15);
myc1->SetRightMargin(0.06);
myc1->SetTopMargin(0.09);
myc1->SetBottomMargin(0.14);
```



Please, take a look to this example:
https://root.cern/doc/master/canvas_8C.html

Example: myfirtsCanvas.C