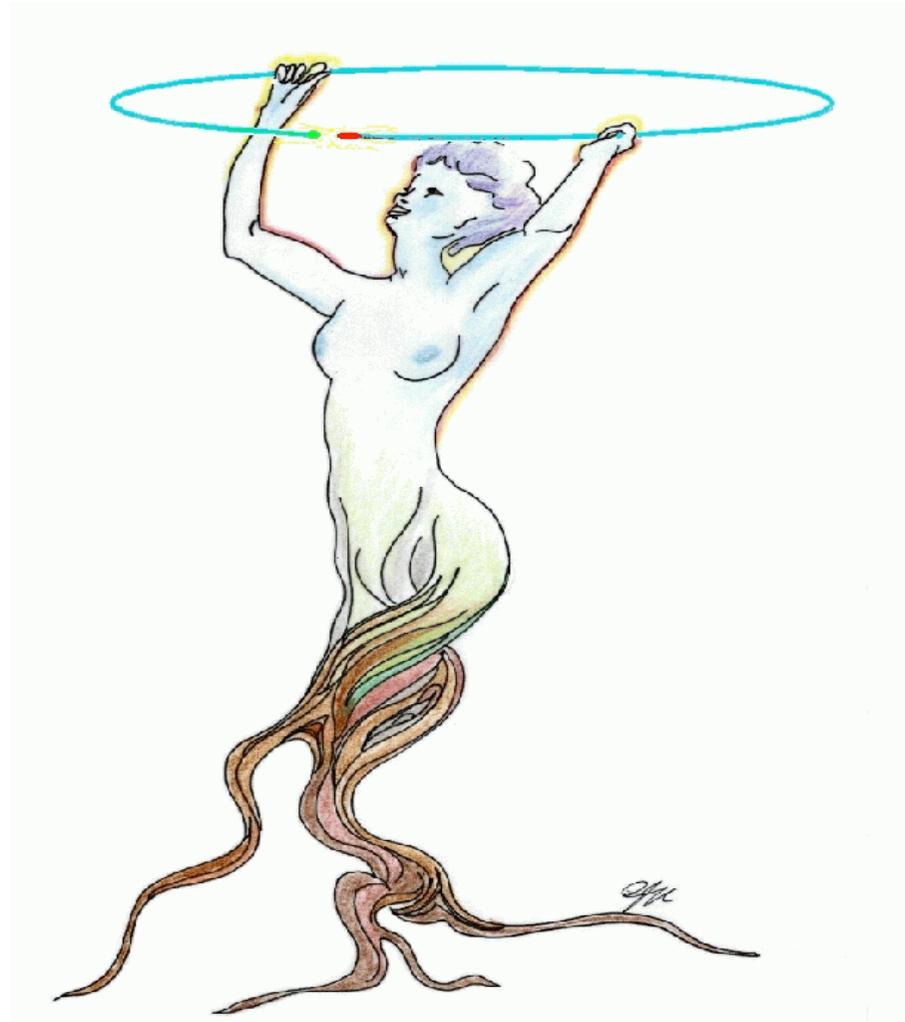


ROOT for beginners

Second Day Programming

Now this won't
hurt a bit...



Writing scripts

- Today:
 - Creation and destruction of objects
 - Manipulating objects
 - Finding the information - the user's guide to the user's guide
 - Finding "lost" objects
 - Writing functions
 - Analysis scripts

Creation and destruction of objects

Commands "new" & "delete"

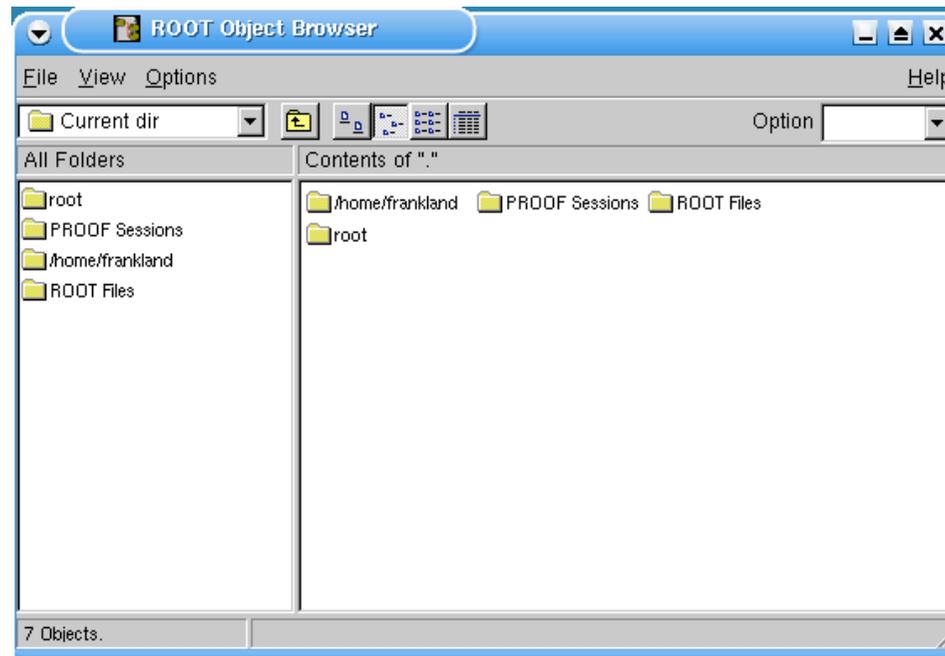
Object pointers

The "new" command

- Yesterday we began the day with:

new TBrowser

which made a "ROOT object browser" appear.



The "new" command

- Yesterday we began the day with:

new TBrowser

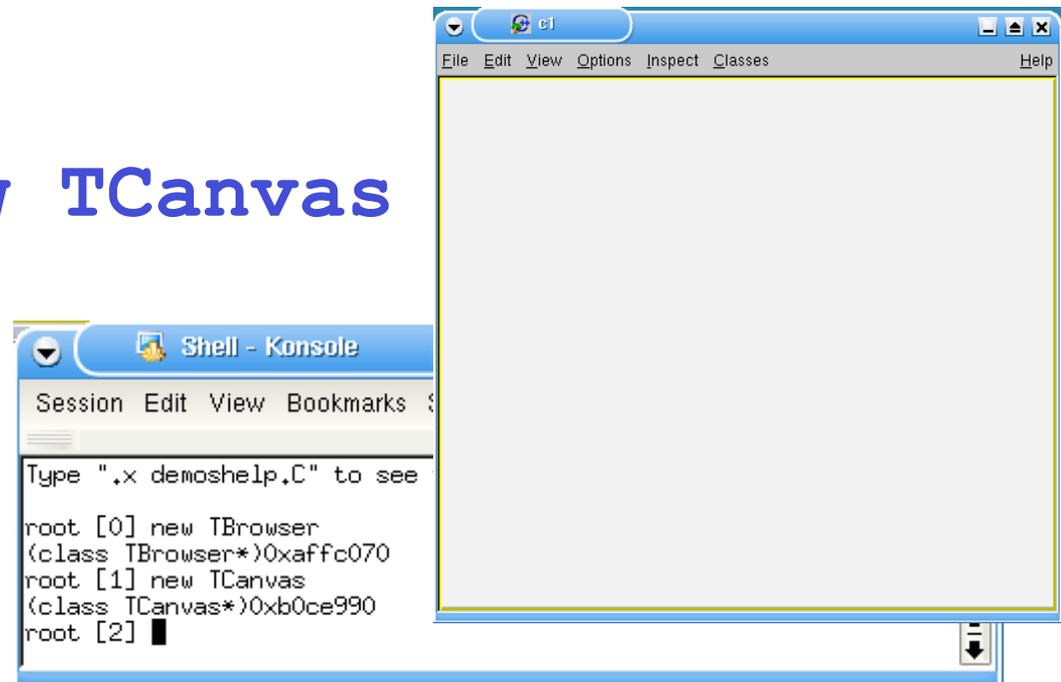
which made a "ROOT object browser" appear.

- If you type

new TCanvas

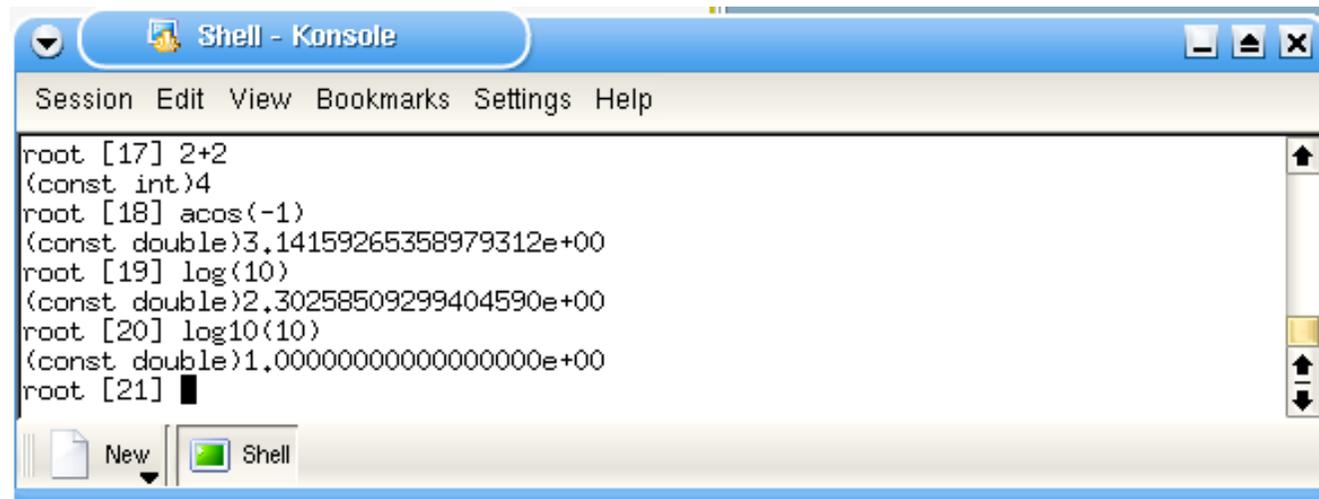
a canvas appears!

But what is going on in the
command window ???



The command window

- The command window is a C++ interpreter!!
- It displays the value of each function, expression or command that you type - try e.g. $2+2$...



```
Shell - Konsole
Session Edit View Bookmarks Settings Help
root [17] 2+2
(const int)4
root [18] acos(-1)
(const double)3.14159265358979312e+00
root [19] log(10)
(const double)2.30258509299404590e+00
root [20] log10(10)
(const double)1.00000000000000000e+00
root [21] █
```

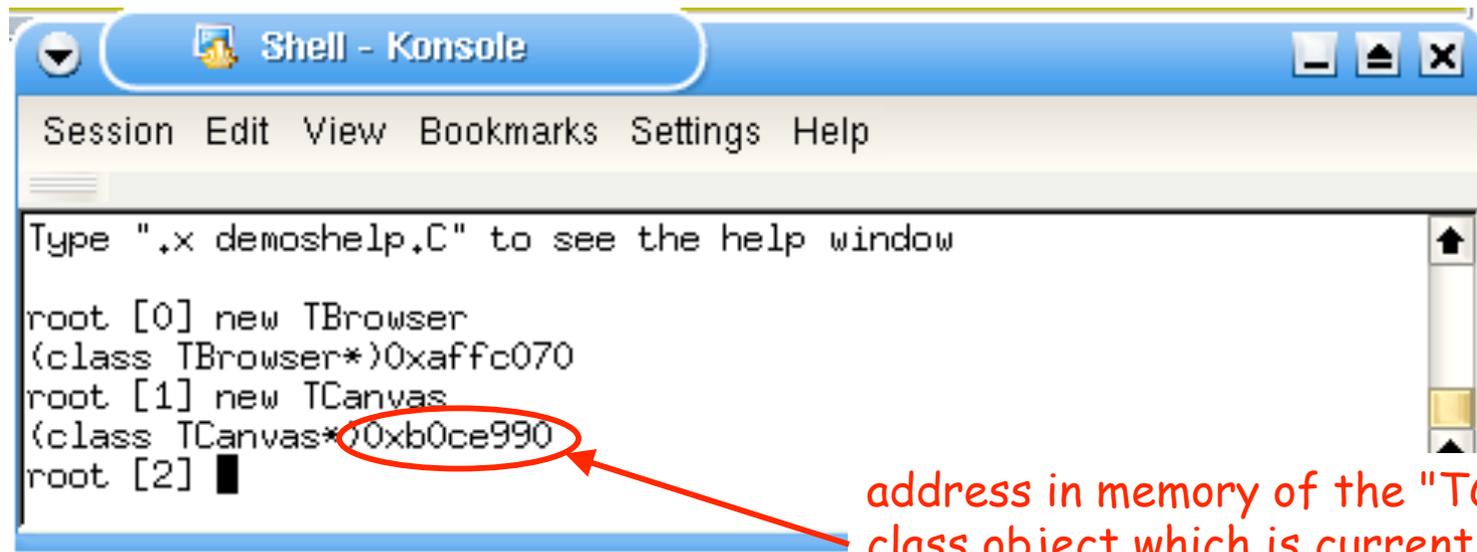
The screenshot shows a window titled "Shell - Konsole" with a menu bar containing "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The main text area displays the following text:

```
root [17] 2+2
(const int)4
root [18] acos(-1)
(const double)3.14159265358979312e+00
root [19] log(10)
(const double)2.30258509299404590e+00
root [20] log10(10)
(const double)1.00000000000000000e+00
root [21] █
```

At the bottom of the window, there is a toolbar with a "New" button and a "Shell" button.

The command window

- The values displayed after a **new** command are the type (*class*) & the address in memory of the created objects



```
Shell - Konsole
Session Edit View Bookmarks Settings Help
Type ".x demoshelp.C" to see the help window
root [0] new TBrowser
(class TBrowser*)0xaffc070
root [1] new TCanvas
(class TCanvas*)0xb0ce990
root [2] █
```

address in memory of the "TCanvas" class object which is currently displayed on your screen

Object pointers

- To use the object, you have to put its address in to a special variable, an *object pointer* (or *pointer*)

Declaration of an
(object) pointer : `ObjectType* toto;`

Object pointers

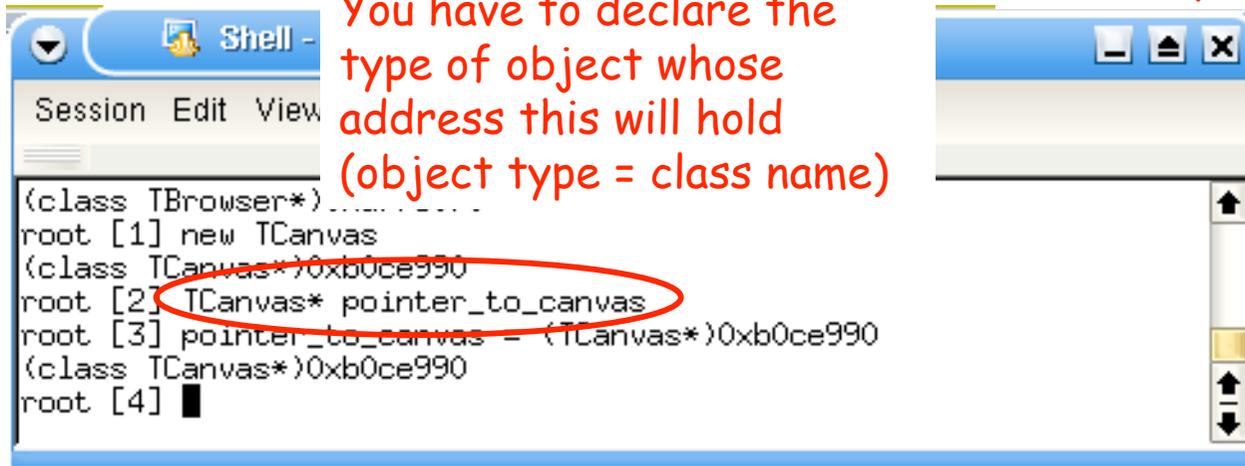
- To use the object, you have to put its address in to a special variable, an *object pointer* (or *pointer*)

Declaration of an
(object) pointer :

ObjectType* toto;

The '*' tells us
this is a pointer

You have to declare the
type of object whose
address this will hold
(object type = class name)



```
(class TBrowser*) .....
root [1] new TCanvas
(class TCanvas*)0xb0ce990
root [2] TCanvas* pointer_to_canvas
root [3] pointer_to_canvas = (TCanvas*)0xb0ce990
(class TCanvas*)0xb0ce990
root [4] █
```

Object pointers

- The value held by the object pointer is the address of the object in memory

Initialisation of an
object pointer :

```
toto = (ObjectType*)address;
```

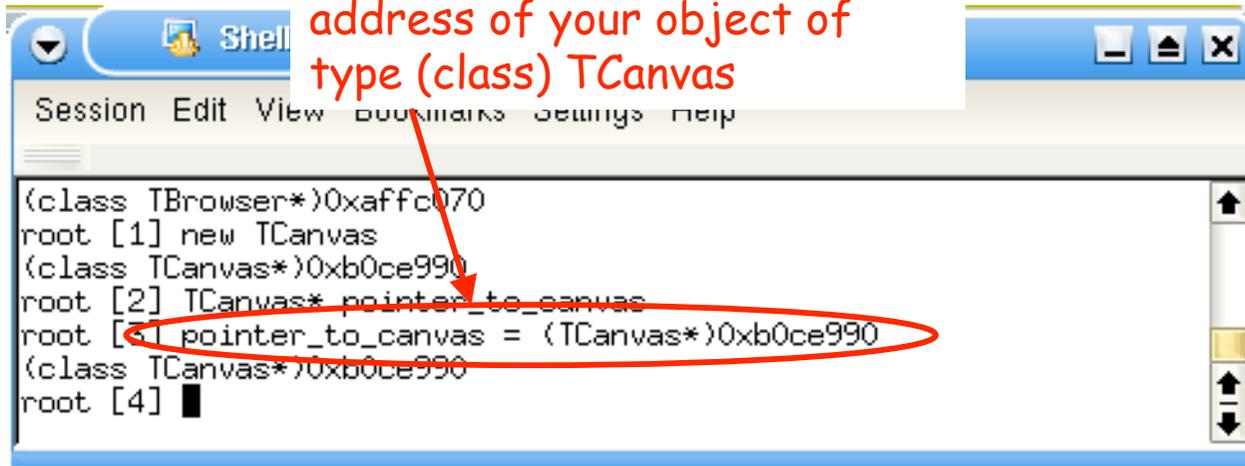
Object pointers

- The value held by the object pointer is the address of the object in memory

Initialisation of an object pointer :

`toto = (ObjectType*)address;`

Initialise the pointer with the address of your object of type (class) TCanvas



```
(class TBrowser*)0xaffc070
root [1] new TCanvas
(class TCanvas*)0xb0ce990
root [2] TCanvas* pointer_to_canvas
root [3] pointer_to_canvas = (TCanvas*)0xb0ce990
(class TCanvas*)0xb0ce990
root [4] █
```

The screenshot shows a terminal window titled 'Shell' with a menu bar containing 'Session Edit View bookmarks settings help'. The terminal output displays the creation of a TCanvas object at memory address 0xb0ce990 and its assignment to a pointer variable named 'pointer_to_canvas'. A red oval highlights the assignment line, and a red arrow points from the explanatory text above to the pointer variable in the code.

Object destruction

- The *delete* command frees the memory space occupied by the objects
- You must use it to destroy all the objects you don't need any more or you will fill the memory!

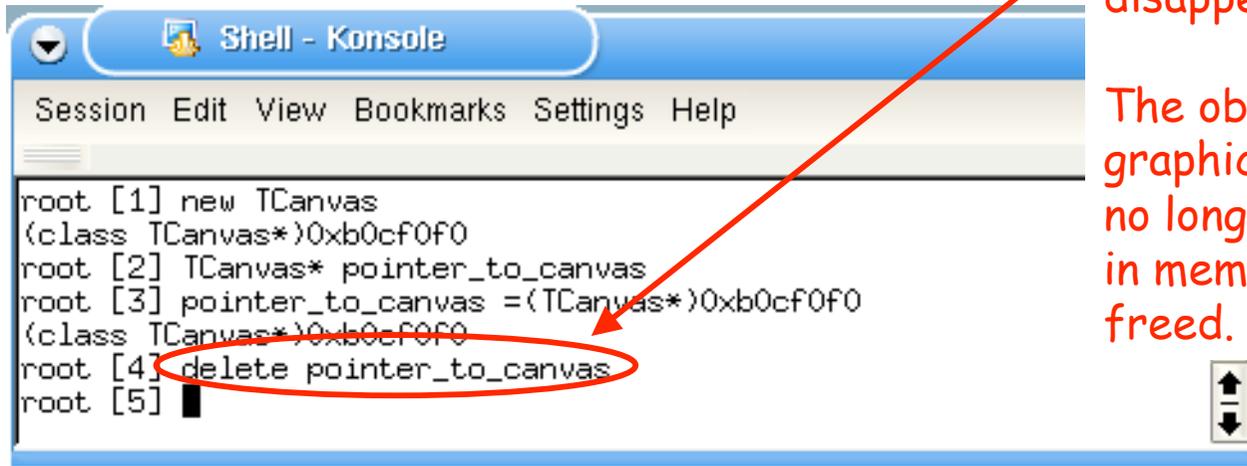
Destroy an object: `delete toto;`

Object destruction

- The *delete* command frees the memory space occupied by the objects
- You must use it to destroy all the objects you don't need any more or you will fill the memory!

Destroy an object: `delete toto;`

Executing this command makes the canvas disappear!



```
root [1] new TCanvas
(class TCanvas*)0xb0cf0f0
root [2] TCanvas* pointer_to_canvas
root [3] pointer_to_canvas =(TCanvas*)0xb0cf0f0
(class TCanvas*)0xb0cf0f0
root [4] delete pointer_to_canvas
root [5] █
```

The object (and its graphical representation) no longer exists, the space in memory it occupied is freed.

Constructors

- Most of the time we will declare a pointer, create an object, and initialise the pointer with the address of the object in one single line !!

Object creation with
declaration and
initialisation of pointer:

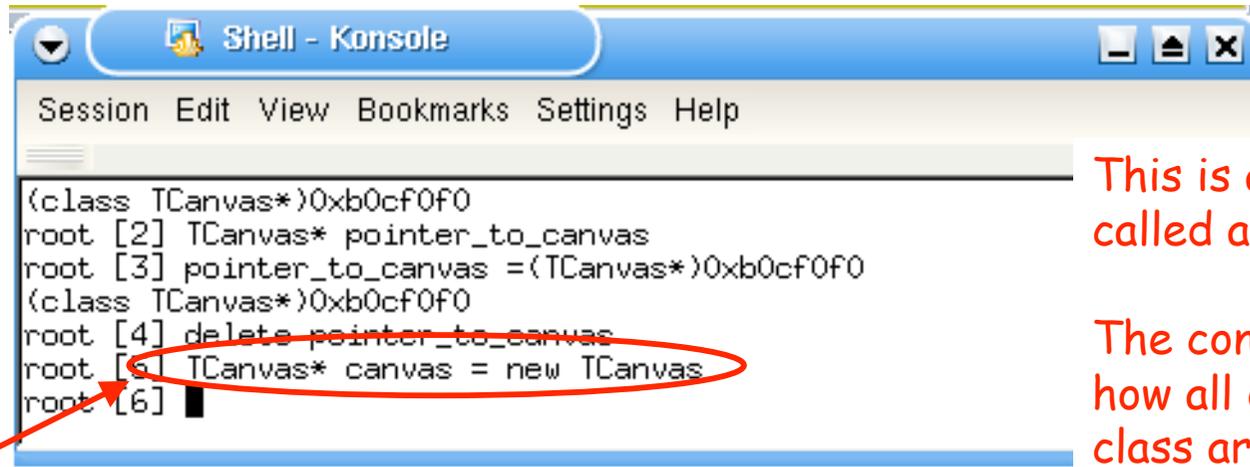
```
ObjectType* toto = new ObjectType;
```

Constructors

- Most of the time we will declare a pointer, create an object, and initialise the pointer with the address of the object in one single line !!

Object creation with declaration and initialisation of pointer:

`ObjectType* toto = new ObjectType;`



```
Shell - Konsole
Session Edit View Bookmarks Settings Help

(class TCanvas*)0xb0cf0f0
root [2] TCanvas* pointer_to_canvas
root [3] pointer_to_canvas =(TCanvas*)0xb0cf0f0
(class TCanvas*)0xb0cf0f0
root [4] delete pointer_to_canvas
root [5] TCanvas* canvas = new TCanvas
root [6]
```

This is a special function called a *constructor*.

The constructor defines how all objects of a given class are made.

*a new canvas appears !

Constructors

- Generally, constructors have arguments

Object creation with
declaration and
initialisation of pointer:

```
ObjectType* toto = new  
ObjectType(...);
```

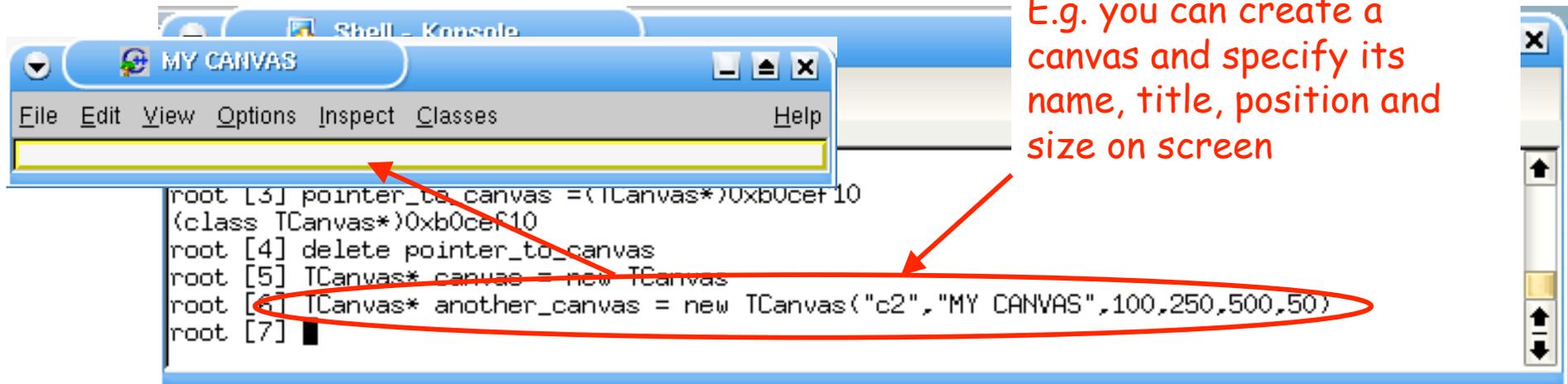
Constructors

- Generally, constructors have arguments

Object creation with
declaration and
initialisation of pointer:

```
ObjectType* toto = new  
ObjectType(...);
```

E.g. you can create a
canvas and specify its
name, title, position and
size on screen



```
root [3] pointer_to_canvas = (TCanvas*)0xb0cef10  
(class TCanvas*)0xb0cef10  
root [4] delete pointer_to_canvas  
root [5] TCanvas* canvas = new TCanvas  
root [6] TCanvas* another_canvas = new TCanvas("c2", "MY CANVAS", 100, 250, 500, 50)  
root [7]
```

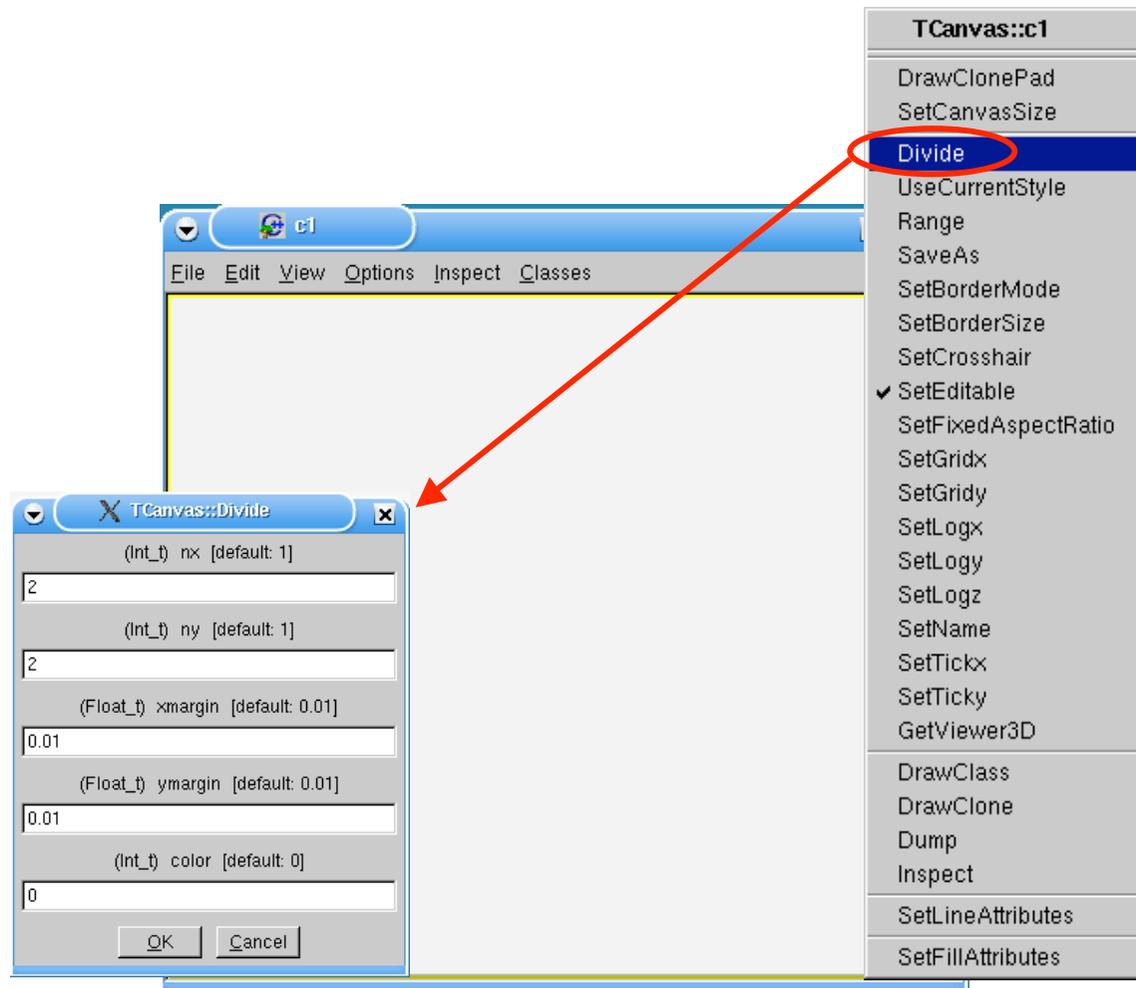
In the previous example, the canvas had
default name "c1" and default title "c1"

Manipulating objects

Getting to grips with class methods

Interacting with objects

- Graphically, we can interact with an object using its context menu:



Example from yesterday, when we divided a canvas in 4

Interacting with objects

- With an object pointer, we can also interact with the object...

Interact with an object using a pointer:

toto->Method(arguments);

The object pointer with the '->' operator define which object we interact with

Interacting with objects

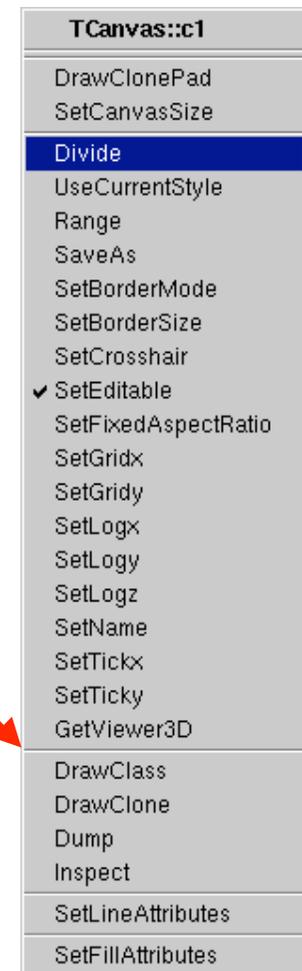
- With an object pointer, we can also interact with the object...

Interact with an object using a pointer:

toto->Method(arguments);

One of the *methods* of the object's class. For example, one of the functions in the context menu.

All objects of the same *class* have the same *methods*.



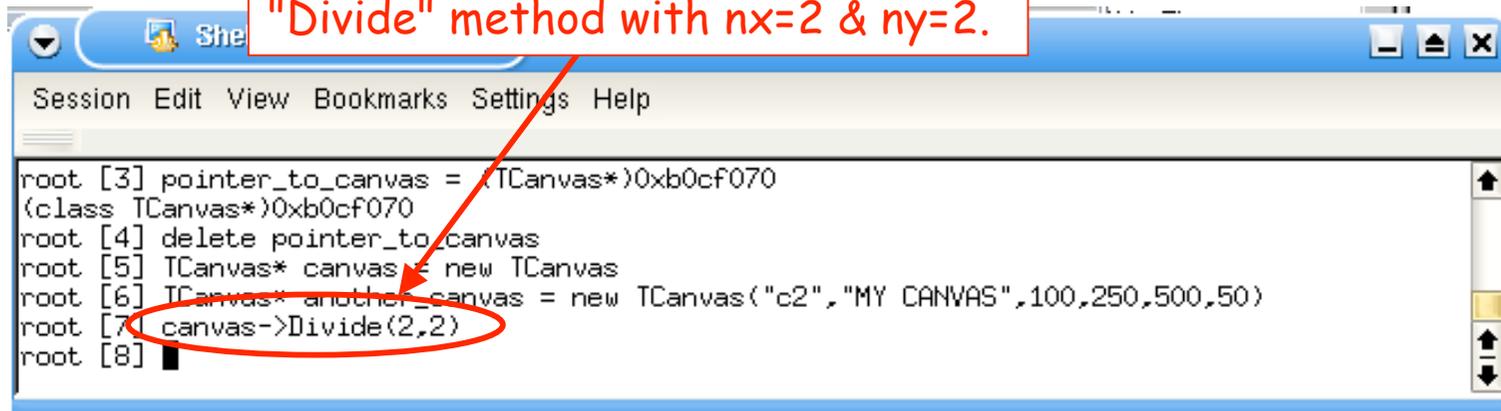
Interacting with objects

- With an object pointer, we can also interact with the object...

Interact with an object using a pointer:

`canvas->Divide(2,2);`

Interact with object "c1" using the pointer "canvas". We use its "Divide" method with nx=2 & ny=2.



```
Session Edit View Bookmarks Settings Help

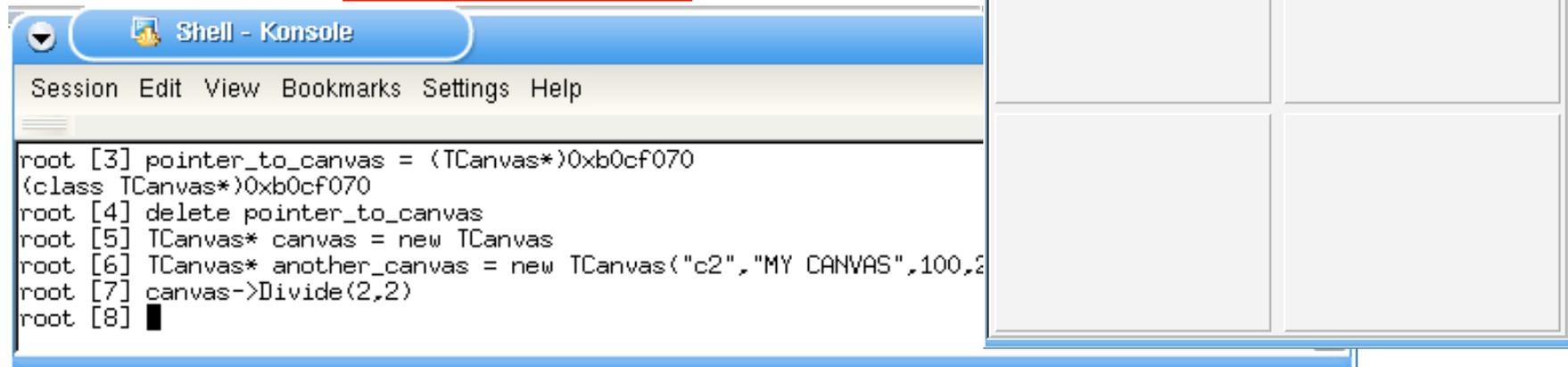
root [3] pointer_to_canvas = (TCanvas*)0xb0cf070
(class TCanvas*)0xb0cf070
root [4] delete pointer_to_canvas
root [5] TCanvas* canvas = new TCanvas
root [6] TCanvas* another_canvas = new TCanvas("c2", "MY CANVAS", 100, 250, 500, 50)
root [7] canvas->Divide(2,2)
root [8]
```

Interacting with objects

- With an object pointer, we can also interact with the object...

Interact with an object using a pointer: `canvas->Divide(2,2);`

The "c1" canvas divides in to 4



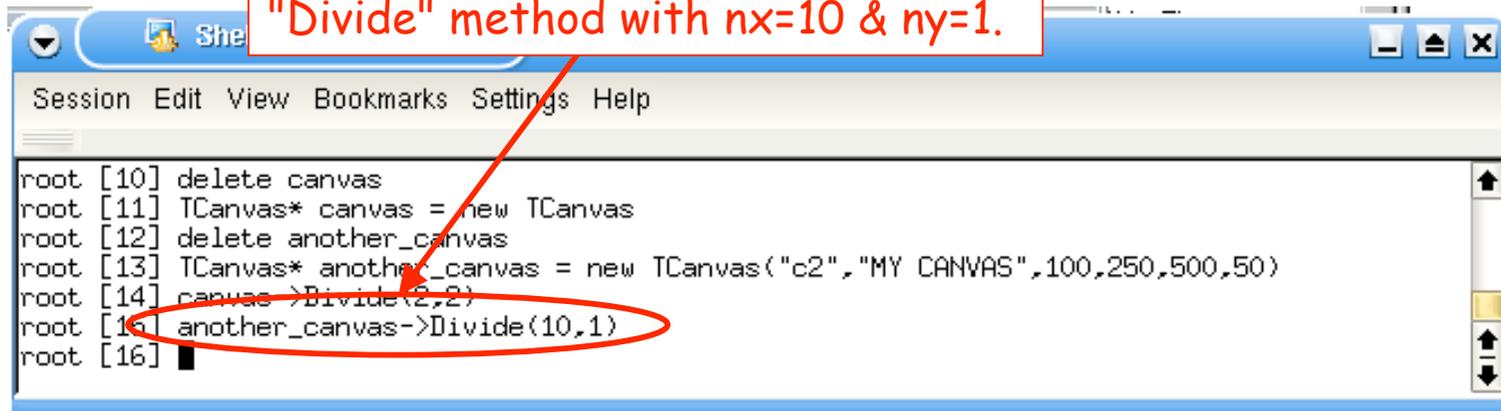
Interacting with objects

- With an object pointer, we can also interact with the object...

Interact with an object using a pointer:

`another_canvas->Divide(10,1);`

Interact with "c2" using pointer "another_canvas". We use c2's "Divide" method with nx=10 & ny=1.



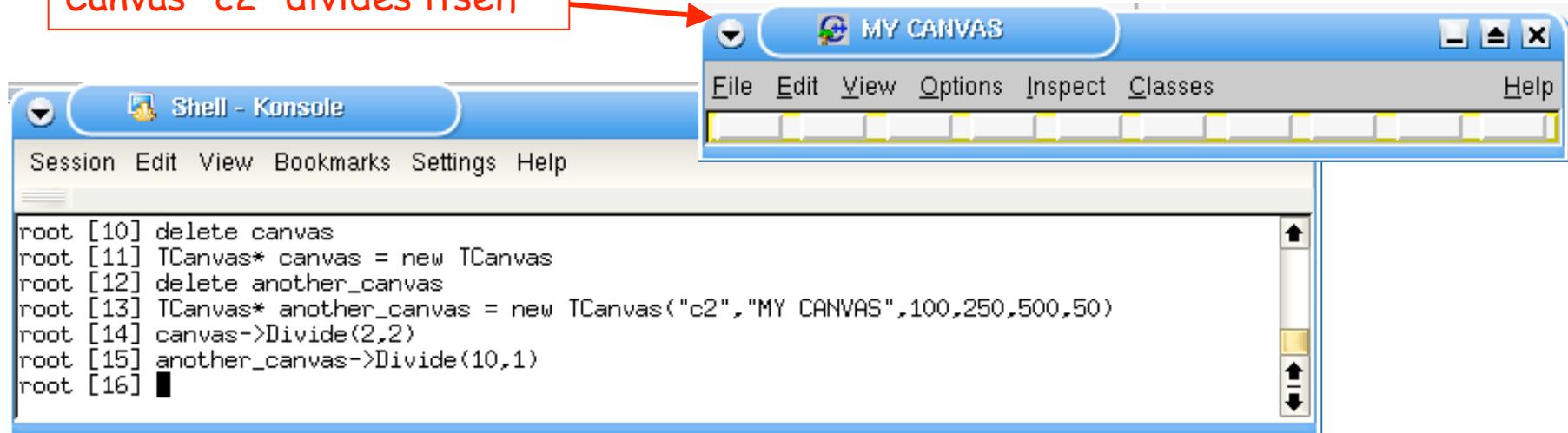
```
Session Edit View Bookmarks Settings Help
root [10] delete canvas
root [11] TCanvas* canvas = new TCanvas
root [12] delete another_canvas
root [13] TCanvas* another_canvas = new TCanvas("c2","MY CANVAS",100,250,500,50)
root [14] canvas->Divide(2,2)
root [15] another_canvas->Divide(10,1)
root [16]
```

Interacting with objects

- With an object pointer, we can also interact with the object...

Interact with an object using a pointer: `another_canvas->Divide(10,1);`

Canvas 'c2' divides itself

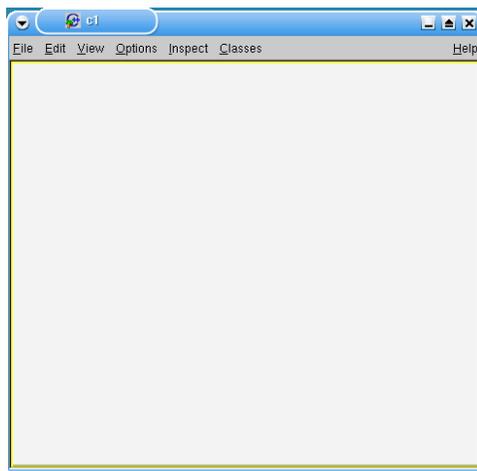
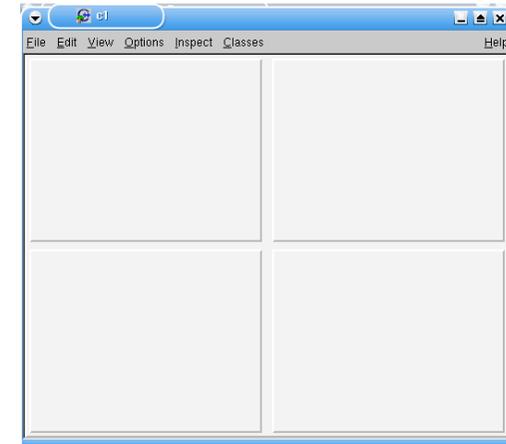


Interacting with objects

- Other operations on canvases:

`canvas->Clear();`

Clear the contents of the canvas, including any divisions



Make a canvas (or a pad) 'active', i.e. its border will become yellow and the next object to be drawn will appear on this canvas

`canvas->cd();`

Example with a histogram

- You can create a 1-D spectrum in much the same way as you create a canvas:

Creating a 1-D
histogram

```
TH1F* histo = new TH1F("h1","My histo", 10, 0., 10.);
```

```
ObjectType* toto = new  
ObjectType(...);
```

N.B. The histogram does not appear
automatically on screen!

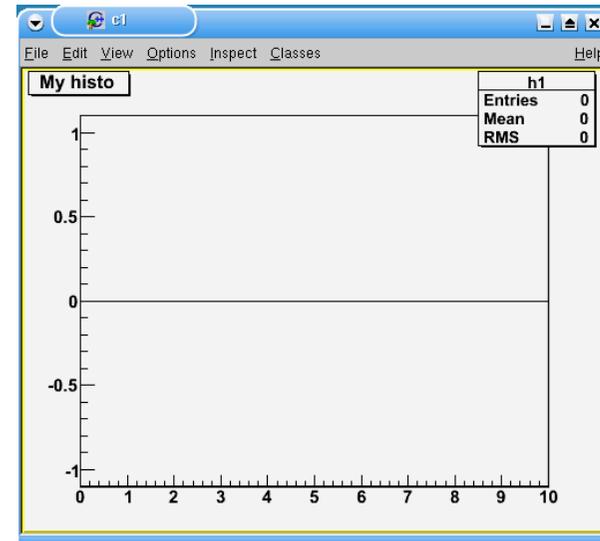
TH1F ? Histogram in 1 dimension
of Floating-point values

Example with a histogram

- Display and fill the histogram:

Display a
histogram:

histo->Draw();



Fill a
histogram:

histo->Fill(3);

The "3" corresponds
to a X-axis value

Nothing seems
to happen ?

Example with a histogram

- Refresh the canvas:

Tell the canvas
that an object
it is displaying
has changed:

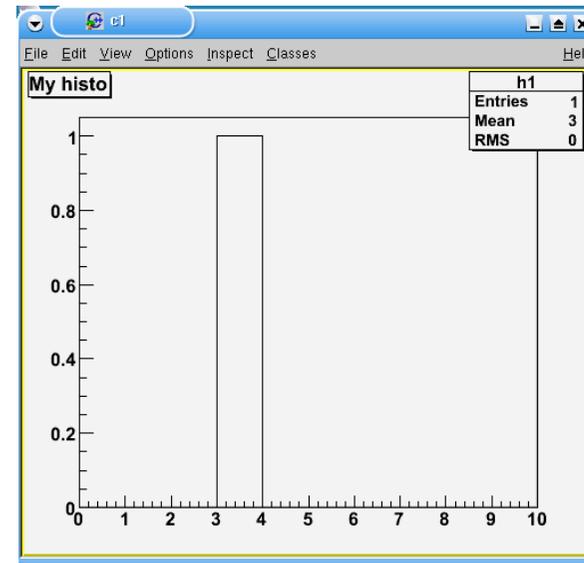
canvas->Modified();

When using the command window, this automatically causes the canvas to refresh.

However, in a script/programme you also have to ask for the canvas to redraw its objects!

Force the
canvas to
refresh:

canvas->Update();



Example with a histogram

- Starting to get bored ?

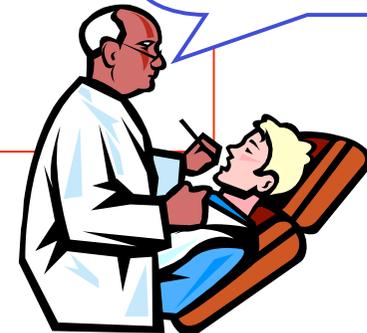
We could carry on like this until we finish filling our histogram...

```
histo->Fill(1.43);  
histo->Fill(6.9);  
...  
histo->Fill(9, 2);  
canvas->Modified();
```

Fill with a weight,
i.e. the value '9'
has occurred
twice

But we'd be happier if we wrote a loop...
...in a function...
...programmed in C++
...!!!

Don't look at
me like that...



Creating objects without "new"

There is another way...

'Temporary' vs. 'Permanent' objects

The other way...

- There is another way to create and manipulate objects...

Creating an object
without "new"

```
ObjectType toto(...);
```

The other way...

- There is another way to create and manipulate objects...

Creating an object
without "new"

ObjectType toto(...);

If the constructor has
arguments, put them here

Creating an
object with "new"

ObjectType* toto_ptr = new ObjectType(...);

The other way...

- There is another way to create and manipulate objects...

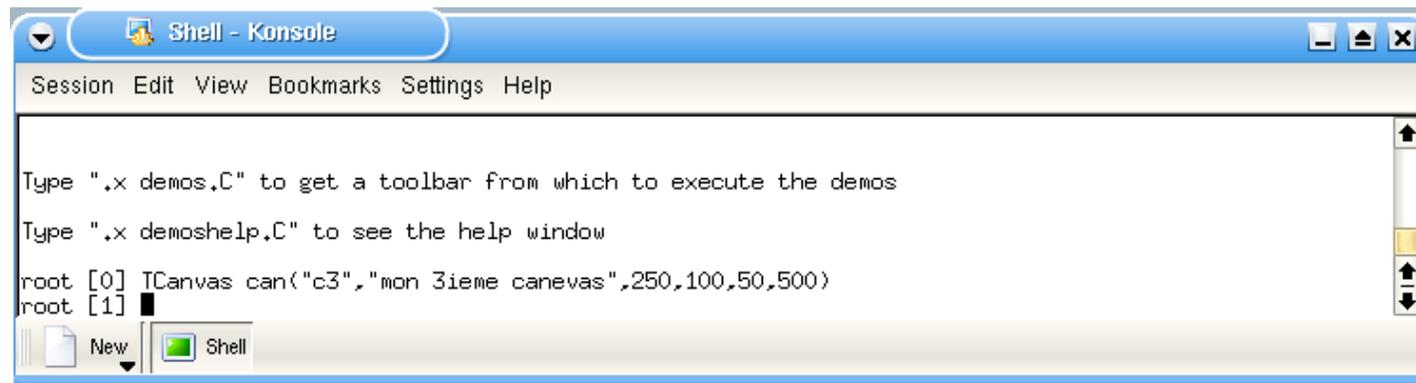
Creating an object
without "new"

```
TCanvas can("c3","title",250,100,50,500);
```



Creating an
object with "new"

```
TCanvas* can_ptr = new TCanvas("c3","title", ...);
```

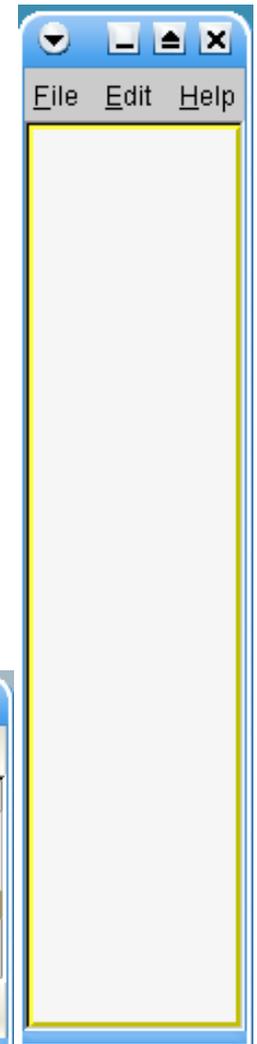
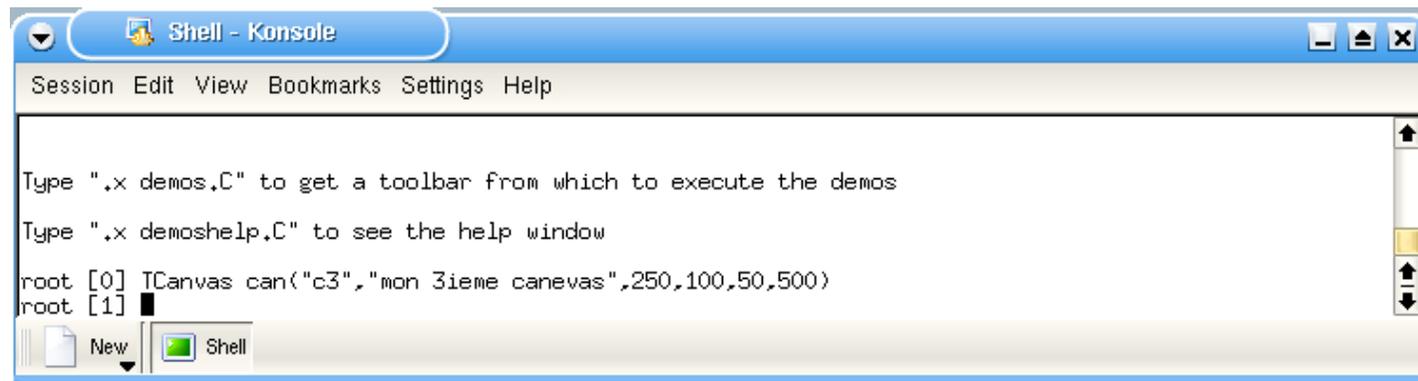


The other way...

- The way we interact with the object isn't quite the same as before...

Interact with an object created without "new"

`toto.Methode(arguments);`



The other way...

- The way we interact with the object isn't quite the same as before...

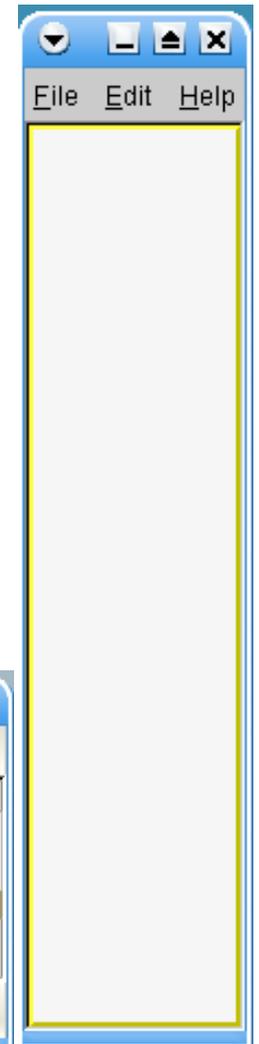
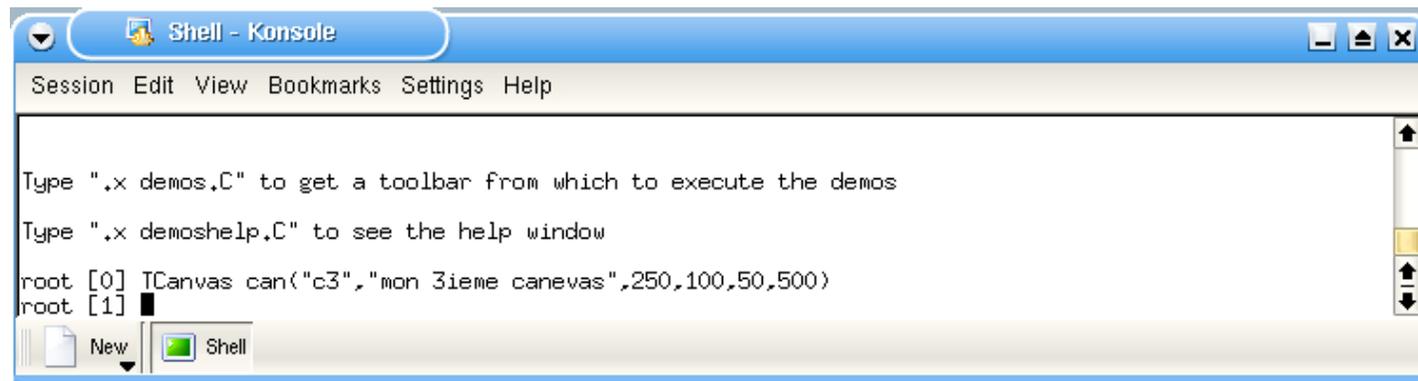
Interact with an object created without "new"

toto.*Methode(arguments);*

In one case we use the '.' operator, in the other '->'

Interact with an object using a pointer:

toto_ptr->*Methode(arguments);*



The other way...

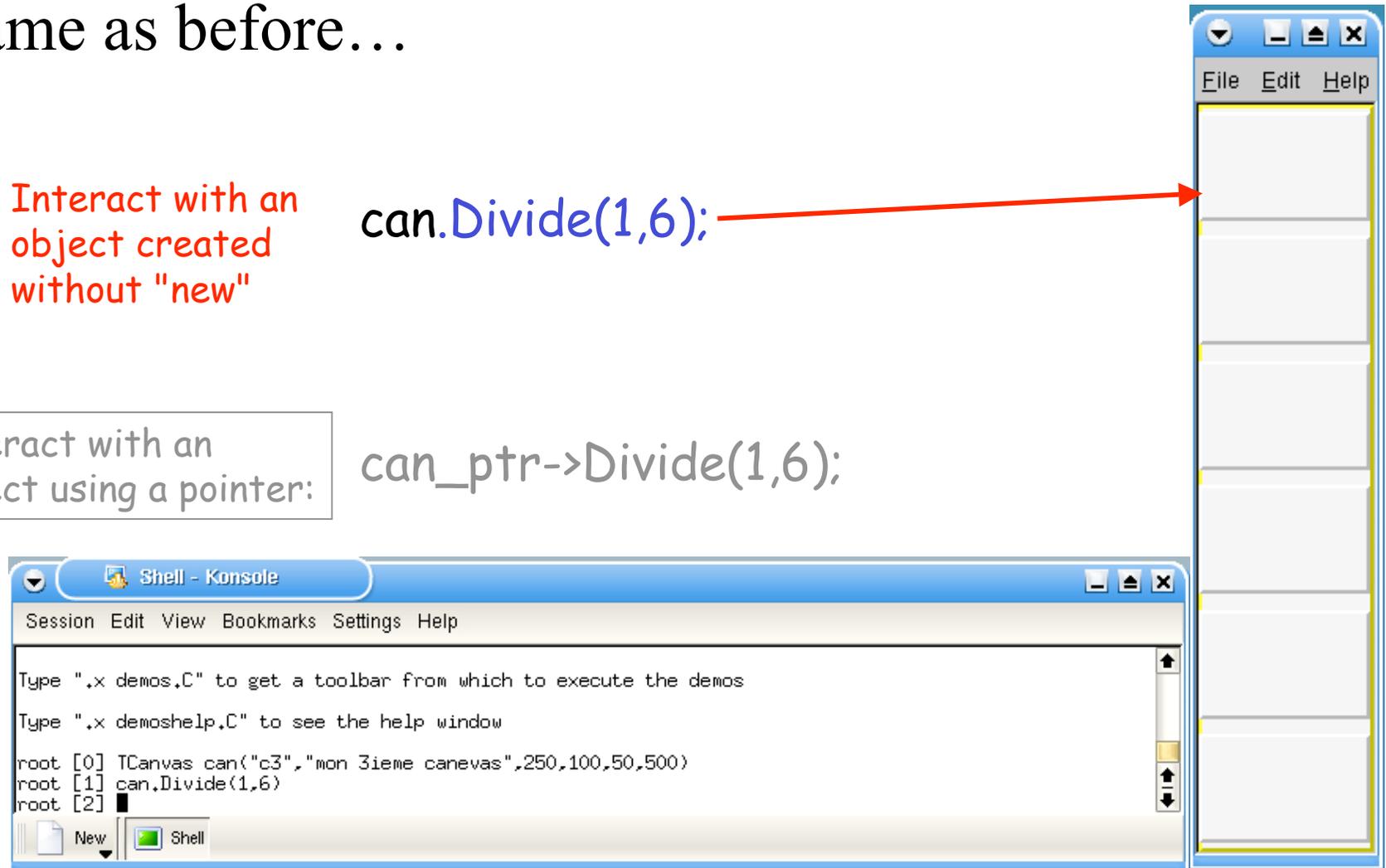
- The way we interact with the object isn't quite the same as before...

Interact with an object created without "new"

`can.Divide(1,6);`

Interact with an object using a pointer:

`can_ptr->Divide(1,6);`



The other way...

- We can also obtain the address in memory of an object created in this way, and then use a *pointer*

Initialise a pointer
with the address of
an existing object

```
ObjectType* toto_ptr = &toto;
```

The other way...

- We can also obtain the address in memory of an object created in this way, and then use a *pointer*

Initialise a pointer with the address of an existing object

```
ObjectType* toto_ptr = &toto;
```

Operator returning the address of the object 'toto'

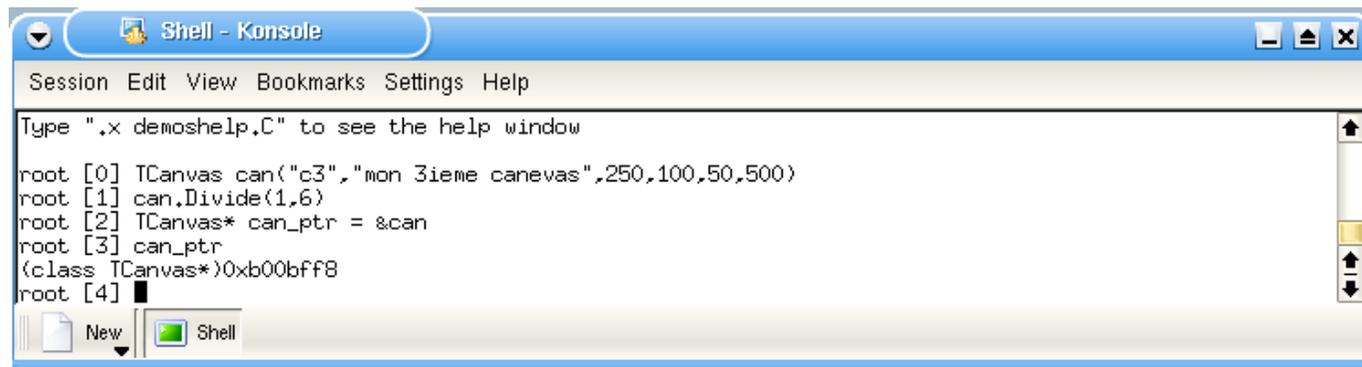
```
Shell - Konsole
Session Edit View Bookmarks Settings Help
Type ".x demoshelp.C" to see the help window
root [0] TCanvas can("c3", "non 3ieme canevas", 250, 100, 50, 500)
root [1] can.Divide(1,6)
root [2] TCanvas* can_ptr = &can
root [3] can_ptr
(class TCanvas*)0xb00bfff8
root [4]
```

The other way...

- The use of a pointer to interact with the object is identical to the previous cases...

Interact with an
object using a pointer

`toto_ptr ->Methode(arguments);`

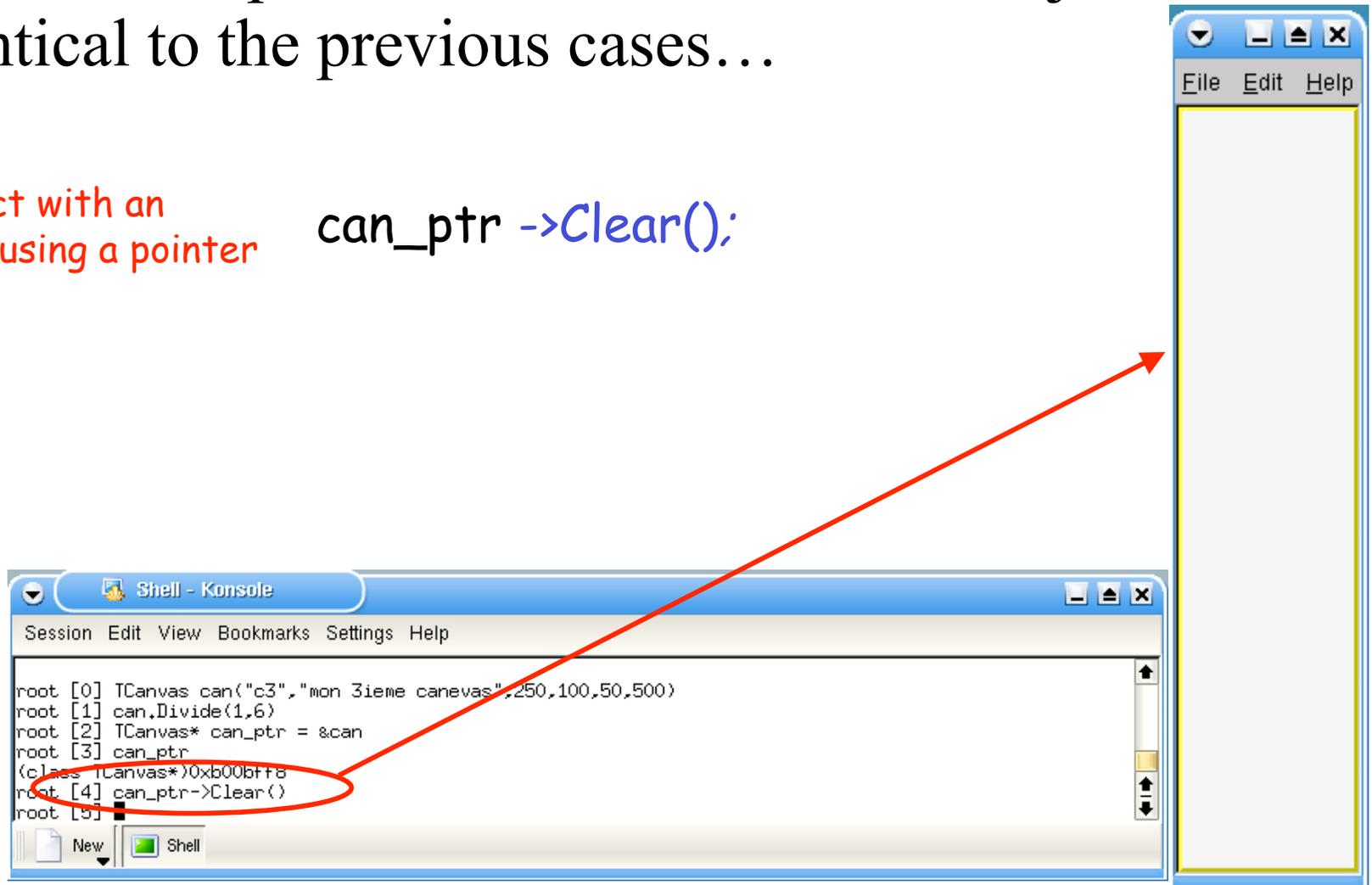


The other way...

- The use of a pointer to interact with the object is identical to the previous cases...

Interact with an object using a pointer

```
can_ptr ->Clear();
```

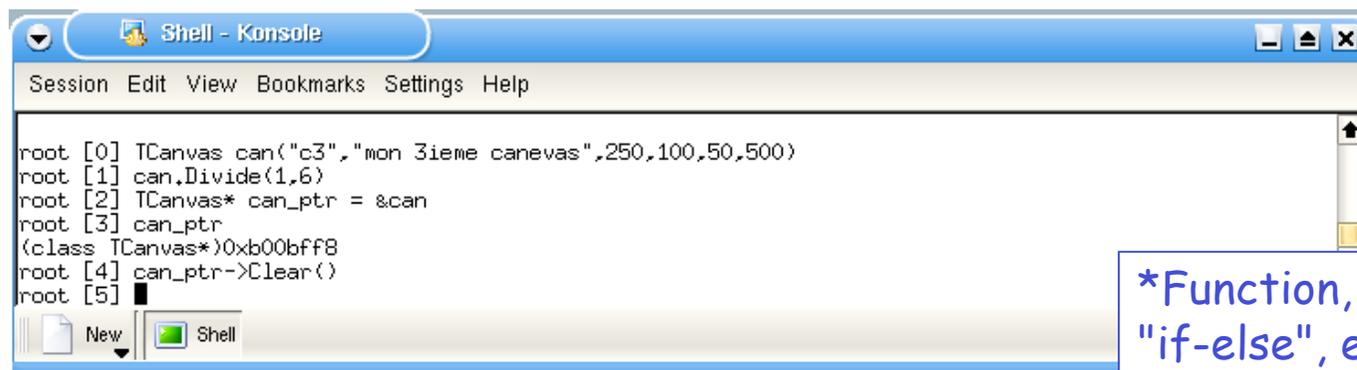


The other way...

- What's the difference ? We don't need to *delete* the object when we've finished with it...

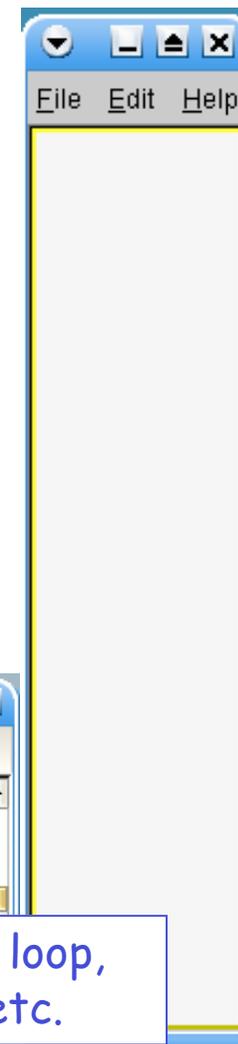
Objects created this way are automatically destroyed at the end of the *code block** in which they are created.

Objects created with "new" are only destroyed by the user, with "delete".



```
root [0] TCanvas can("c3", "mon 3ieme canevas", 250, 100, 50, 500)
root [1] can.Divide(1,6)
root [2] TCanvas* can_ptr = &can
root [3] can_ptr
(class TCanvas*)0xb00bfff8
root [4] can_ptr->Clear()
root [5] █
```

*Function, loop, "if-else", etc.



The other way...

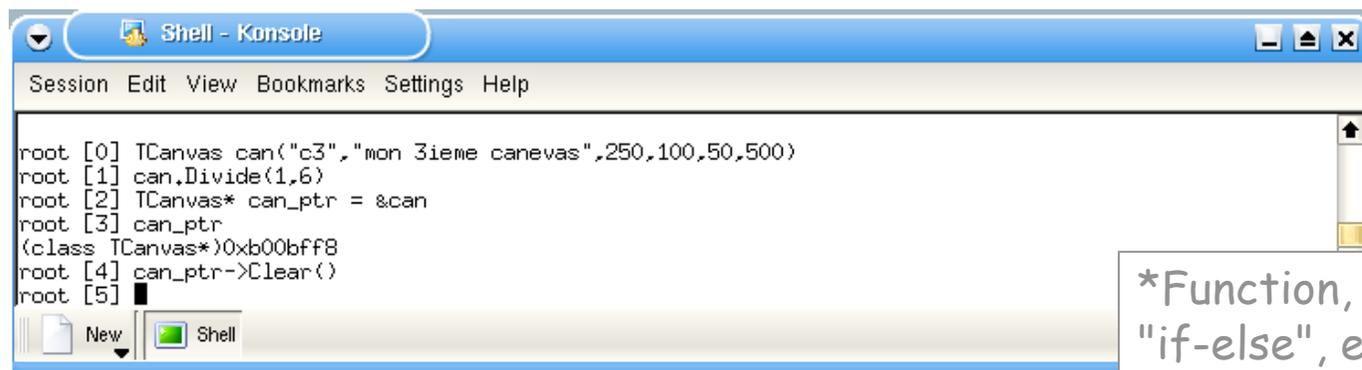
- What's the difference ? We don't need to *delete* the object when we've finished with it...

Temporary
objects

Objects created this way are automatically destroyed at the end of the *code block** in which they are created.

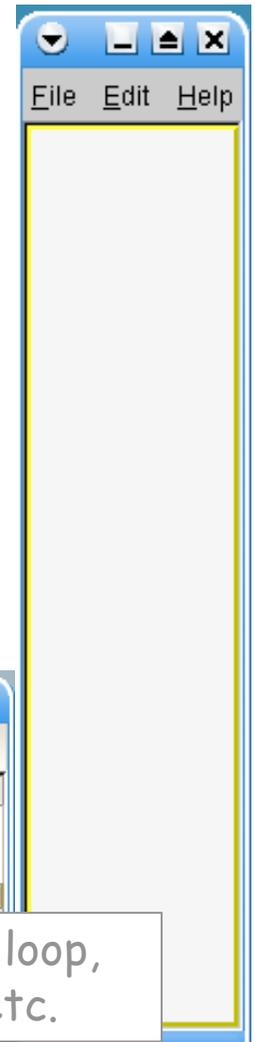
Permanent
objects

Objects created with "new" are only destroyed by the user, with "delete".



```
root [0] TCanvas can("c3", "mon 3ieme canevas", 250, 100, 50, 500)
root [1] can.Divide(1,6)
root [2] TCanvas* can_ptr = &can
root [3] can_ptr
(class TCanvas*)0xb00bfff8
root [4] can_ptr->Clear()
root [5] █
```

*Function, loop,
"if-else", etc.



Getting information on classes

Where's the manual ?

Where's the manual for the manual ?

Do I have to learn it all by heart ?*

*NO!!!!

Getting to know your way around

- How can I find out all the possible ways to interact with an object ?
- How do I find out all the methods of a class ?

1. command line completion with <TAB>

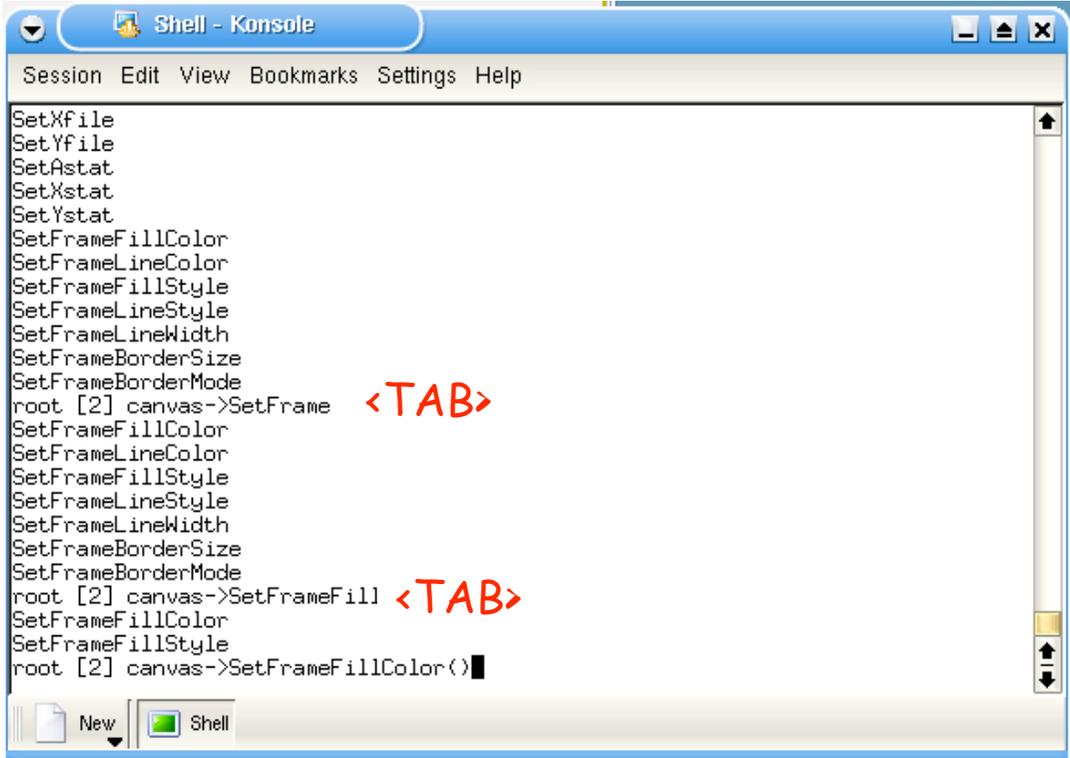
Very efficient, reduces to bare minimum the amount you have to type (and so reduces typing errors...)

TIP 1:

most methods which change an object begin "Set..."

TIP 2:

most methods which give an information about an object begin "Get..."



```
Shell - Konsole
Session Edit View Bookmarks Settings Help
SetXfile
SetYfile
SetAstat
SetXstat
SetYstat
SetFrameFillColor
SetFrameLineColor
SetFrameFillStyle
SetFrameLineStyle
SetFrameLineWidth
SetFrameBorderSize
SetFrameBorderMode
root [2] canvas->SetFrame <TAB>
SetFrameFillColor
SetFrameLineColor
SetFrameFillStyle
SetFrameLineStyle
SetFrameLineWidth
SetFrameBorderSize
SetFrameBorderMode
root [2] canvas->SetFrameFill <TAB>
SetFrameFillColor
SetFrameFillStyle
root [2] canvas->SetFrameFillColor()
```

Getting to know your way around

- The best method: <http://root.cern.ch>

2. Look at the ROOT web site

1. click on "Reference Guide"

The screenshot shows the ROOT System Home Page in a Konqueror browser window. The browser's address bar displays <http://root.cern.ch/>. The page features a navigation menu on the left with items such as Roadmap, Mission Statement, Architecture, Main Features, CINT, Coding Conventions, Benchmarking, Picture Gallery, Publication List, The ROOT Team, License, Register as User, Download Binaries, Install from Source, CVS, ViewCVS, LXR, User's Guide, Reference Guide, Tutorials, HOWTO's, RootTalk Forum, RootTalk Digest, Example Applications, BaBar Tutorials, FNAL Tutorials, MINOS Tutorials, PROOF, Report a Bug, Feedback, Search, and News. The 'Reference Guide' link is circled in red, with a red arrow pointing to it from the text '1. click on "Reference Guide"'. The main content area displays the ROOT logo, the text 'An Object-Oriented Data Analysis Framework', and an illustration of a woman holding a glowing orb. A green banner at the bottom of the page announces 'Production Version 4.04/02 NEW' dated '04/05/2005'. Below the banner, it states 'The Production release of ROOT 4.04/02 is now available.' and provides a link to 'THIS IMPORTANT ANNOUNCEMENT.'. It also mentions 'See also the announcement of the ROOT2005 Workshop.' and lists 'Tar files for the source, documentation and binaries are available at: Version 4.04/02 Release Notes http://root.cern.ch/root/Version404.html'. At the bottom, it notes 'The CVS tag for this version is v4-04-02.'

Getting to know your way around

- The best method: <http://root.cern.ch>

2. list of classes by category (histos, matrices, 3D geometry, etc.)

ROOT Reference Guide - Konqueror

Location Edit View Go Bookmarks Tools Settings Window Help

Location: <http://root.cern.ch/root/Reference.html>

ROOT Reference Guide

The ROOT reference guide is automatically generated by the [ROOT documentation system](#).

You can browse the reference guide either per "Class Category" or via a single list of all classes:

- [The ROOT Class Categories](#)
- [Classes and Members Reference Guide \(Development version 5 in CVS\)](#)
- [Classes and Members Reference Guide \(pro version 4.04/02\)](#)
- [Classes and Members Reference Guide \(pro version 4.00/08\)](#)
- [Classes and Members Reference Guide \(pro version 3.10/02\)](#)
- [Classes and Members Reference Guide \(old version 3.05/07\)](#)
- [Classes and Members Reference Guide \(old version 3.04\)](#)
- [Classes and Members Reference Guide \(old version 3.03\)](#)
- [Classes and Members Reference Guide \(old version 3.02/07\)](#)
- [Classes and Members Reference Guide \(old version 3.01/06\)](#)
- [Classes and Members Reference Guide \(very old version 3.00\)](#)
- [Classes and Members Reference Guide \(ultra old version 2.25\)](#)
- [Cross-Reference of Sources in CVS using LXR](#)
- [Browsing the CVS repository using ViewCVS](#)

Use this field for a quick search of the reference guide:

For a more complete access to the search engine see the [search](#) page.

Rene Brun, Fons Rademakers
Last update 19/07/2004 by RB

Getting to know your way around

- The best method: <http://root.cern.ch>

2. complete list of all classes for each version of ROOT

ROOT Reference Guide - Konqueror

Location Edit View Go Bookmarks Tools Settings Window Help

Location: <http://root.cern.ch/root/Reference.html>

ROOT Reference Guide

The ROOT reference guide is automatically generated by the ROOT [documentation system](#).

You can browse the reference guide either per "Class Category" or via a single list of all classes:

[The ROOT Class Categories](#)

- [Classes and Members Reference Guide \(Development version 5 in CVS\)](#)
- [Classes and Members Reference Guide \(pro version 4.04/02\)](#)
- [Classes and Members Reference Guide \(pro version 4.00/08\)](#)
- [Classes and Members Reference Guide \(pro version 3.10/02\)](#)
- [Classes and Members Reference Guide \(old version 3.05/07\)](#)
- [Classes and Members Reference Guide \(old version 3.04\)](#)
- [Classes and Members Reference Guide \(old version 3.03\)](#)
- [Classes and Members Reference Guide \(old version 3.02/07\)](#)
- [Classes and Members Reference Guide \(old version 3.01/06\)](#)
- [Classes and Members Reference Guide \(very old version 3.00\)](#)
- [Classes and Members Reference Guide \(ultra old version 2.25\)](#)
- [Cross Reference of Sources in CVS using LXR](#)
- [Browsing the CVS repository using ViewCVS](#)

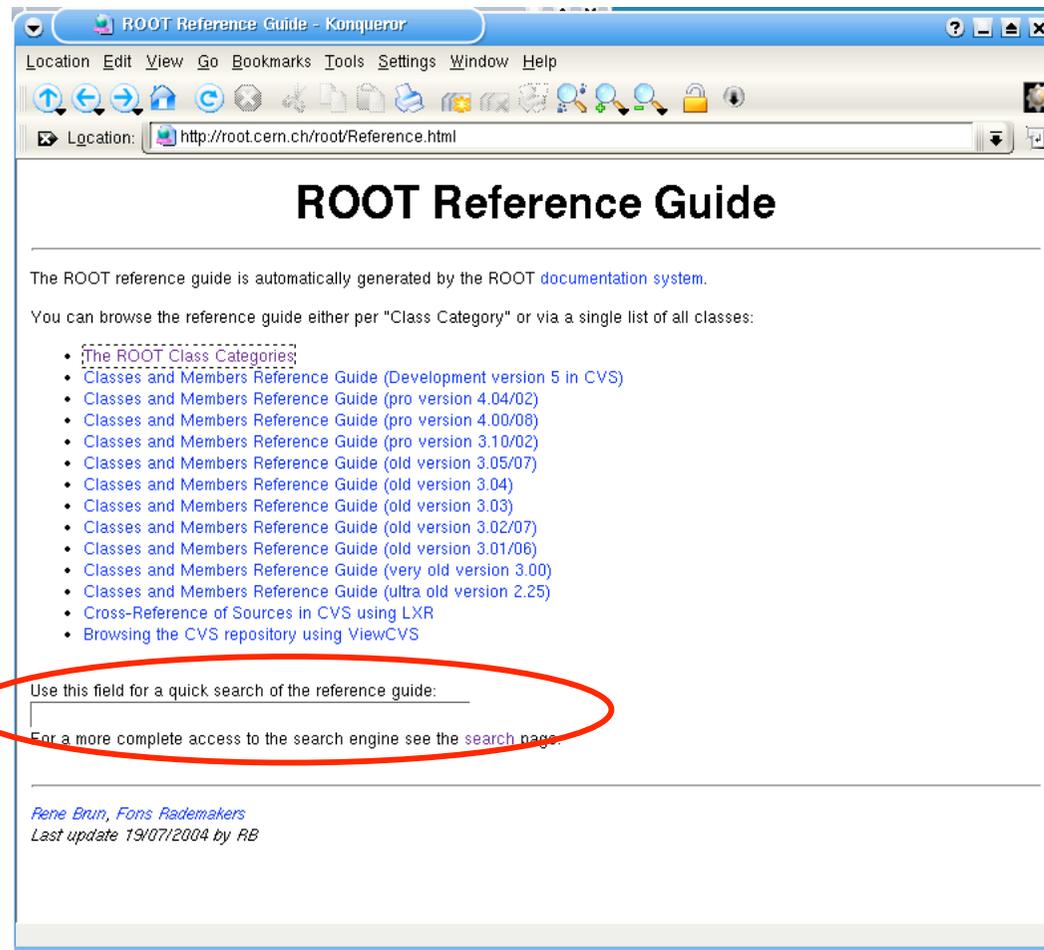
Use this field for a quick search of the reference guide:

For a more complete access to the search engine see the [search](#) page.

Rene Brun, Fons Rademakers
Last update 19/07/2004 by RB

Getting to know your way around

- The best method: <http://root.cern.ch>



2. keyword search

Getting to know your way around

- The best method: <http://root.cern.ch>

type "TCanvas"
then <ENTER>/
<RETURN>

ROOT Reference Guide - Konqueror

Location Edit View Go Bookmarks Tools Settings Window Help

Location: <http://root.cern.ch/root/Reference.html>

ROOT Reference Guide

The ROOT reference guide is automatically generated by the ROOT [documentation system](#).

You can browse the reference guide either per "Class Category" or via a single list of all classes:

- [The ROOT Class Categories](#)
- [Classes and Members Reference Guide \(Development version 5 in CVS\)](#)
- [Classes and Members Reference Guide \(pro version 4.04/02\)](#)
- [Classes and Members Reference Guide \(pro version 4.00/08\)](#)
- [Classes and Members Reference Guide \(pro version 3.10/02\)](#)
- [Classes and Members Reference Guide \(old version 3.05/07\)](#)
- [Classes and Members Reference Guide \(old version 3.04\)](#)
- [Classes and Members Reference Guide \(old version 3.03\)](#)
- [Classes and Members Reference Guide \(old version 3.02/07\)](#)
- [Classes and Members Reference Guide \(old version 3.01/06\)](#)
- [Classes and Members Reference Guide \(very old version 3.00\)](#)
- [Classes and Members Reference Guide \(ultra old version 2.25\)](#)
- [Cross-Reference of Sources in CVS using LXR](#)
- [Browsing the CVS repository using ViewCVS](#)

Use this field for a quick search of the reference guide:

For a more complete access to the search engine see the [search page](#).

Rene Brun, Fons Rademakers
Last update 19/07/2004 by RB

Getting to know your way around

- The best method: `http://root.cern.ch`

searching using the name of a class works best...

...but don't neglect the other responses which can be very interesting!!

Page	File Size	Score
TCanvas	70568	1000
TDialogCanvas	19524	714
TInspectCanvas	18202	714
TCanvas - source file	116199	714
A simple fitting example	3349	428
Simple Formula and Functions	3168	428
create a canvas and save as png	2309	428
TInspectorImp - source file	3106	428
Index of GPAD classes	5270	285
simple example showing the GUI signal/slots mechanism	3158	285
Filling histograms with random numbers from a function	5063	285
A Simple Fitting Example	4615	285
An Example of Object Oriented User Interface	5189	285
illustration of the TASImage class and an image editor	3113	285
Examples of a Graph with error bars	3227	285
A simple graph with axis titles	4185	285
TGraph2DErrors example	3736	285
Using the TLatex class	2323	285
Editing with a user defined function	3604	285

Presentation of a page documenting a class

- All necessary information is here... as long as you know where to look

Class name and
name of its closest
family relation
(parent class)

class **TCanvas** : public **TPad**

Inheritance Chart:

```
graph TD
    TCanvas --> TPad
    TPad --> TVirtualPad
    TVirtualPad --> TAttFill
    TAttFill --> TAttLine
    TAttLine --> TObject
    TAttLine --> TAttMarkerCanvas
    TAttLine --> TAttTextCanvas
    TAttLine --> TDialogCanvas
    TAttLine --> TInspectCanvas
    TAttTextCanvas --> TDrawPanelHist
    TDrawPanelHist --> TFitPanel
    TFitPanel --> TFitPanelGraph
```

private:

```
TCanvas(const TCanvas& canvas)
void Build()
virtual void CopyPixmaps()
void DrawEventStatus(Int_t event, Int_t x, Int_t y, TObject* selected)
TCanvas& operator=(const TCanvas& rhs)
void DrawAutoExec()
```

Presentation of a page documenting a class

- All necessary information is here... as long as you know where to look

Complete family tree for the class

library: libGpad
#include "TCanvas.h"

TCanvas

[class description - source file - inheritance tree \(.pdf\)](#)

class TCanvas : public TPad

Inheritance Chart:

```
graph TD
    TCanvas --> TPad
    TCanvas --> TVirtualPad
    TCanvas --> TAttFill
    TCanvas --> TAttLine
    TCanvas --> TAttPad
    TCanvas --> TQObject
    TCanvas --> TAttFillCanvas
    TCanvas --> TAttLineCanvas
    TCanvas --> TAttMarkerCanvas
    TCanvas --> TAttTextCanvas
    TCanvas --> TDialogCanvas
    TCanvas --> TDrawPanelHist
    TCanvas --> TFitPanel
    TCanvas --> TFitPanelGraph
    TCanvas --> TInspectCanvas
```

private:

```
TCanvas(const TCanvas& canvas)
void Build()
virtual void CopyPixmaps()
void DrawEventStatus(Int_t event, Int_t x, Int_t y, TObject* selected)
TCanvas& operator=(const TCanvas& rhs)
void DrawAutoExec()
```

Presentation of a page documenting a class

- All necessary information is here... as long as you know where to look

Complete family tree of the class

parents and grand-parents to the left

TCanvas

library: libGpad
#include "TCanvas.h"

[class description - source file - inheritance tree \(.pdf\)](#)

class TCanvas : public TPad

Inheritance Chart:

```
graph TD
    TCanvas --> TPad
    TPad --> TVirtualPad
    TPad --> TAttFill
    TPad --> TAttLine
    TPad --> TAttText
    TPad --> TAttMarker
    TPad --> TDialogCanvas
    TPad --> TDrawPanelHist
    TPad --> TFitPanel
    TFitPanel --> TFitPanelGraph
```

private:

```
TCanvas(const TCanvas& canvas)
void Build()
virtual void CopyPixmaps()
void DrawEventStatus(Int_t event, Int_t x, Int_t y, TObject* selected)
TCanvas& operator=(const TCanvas& rhs)
void DrawAutoExec()
```

Presentation of a page documenting a class

- All necessary information is here... as long as you know where to look

Complete family tree of the class

parents and grand-parents to the left

children and grand-children to the right

TCanvas

library: libGpad
#include "TCanvas.h"

[class description](#) - [source file](#) - [inheritance tree \(.pdf\)](#)

```
class TCanvas : public TPad
```

Inheritance Chart:

```
graph TD
    TCanvas --> TDialogCanvas
    TCanvas --> TAttFillCanvas
    TCanvas --> TAttLineCanvas
    TCanvas --> TAttMarkerCanvas
    TCanvas --> TAttTextCanvas
    TCanvas --> TDrawPanelHist
    TCanvas --> TFitPanel
    TFitPanel --> TFitPanelGraph
    TInspectCanvas
```

```
private:
    TCanvas(const TCanvas& canvas)
    void Build()
    virtual void CopyPixmaps()
    void DrawEventStatus(Int_t event, Int_t x, Int_t y, TObject* selected)
    TCanvas& operator=(const TCanvas& rhs)
    void DrawAutoExec()
```

Presentation of a page documenting a class

- All necessary information is here... as long as you know where to look

(a bit further down)

complete list of class methods

each method name is a link to an explanation of the method



```
private:
    TCanvas(const TCanvas& canvas)
    void Build()
    virtual void CopyPixmaps()
    void DrawEventStatus(Int_t event, Int_t x, Int_t y, TObject* selected)
    TCanvas& operator=(const TCanvas& rhs)
    void RunAutoExec()

protected:
    void Constructor()
    void Constructor(const char* name, const char* title, Int_t form)
    void Constructor(const char* name, const char* title, Int_t ww, Int_t wh)
    void Constructor(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t wright, Int_t wbottom)
    void Destructor()
    virtual void ExecuteEvent(Int_t event, Int_t px, Int_t py)
    void Init()

public:
    TCanvas(Bool_t build = kTRUE)
    TCanvas(const char* name, const char* title = "", Int_t form = 1)
    TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
    TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t wright, Int_t wbottom)
    TCanvas(const char* name, Int_t ww, Int_t wh, Int_t winid)

    virtual ~TCanvas()
    virtual void Browse(TBrowser* b)
    virtual TVirtualPad* cd(Int_t subpadnumber = 0)
    static TClass* Class()
    virtual void Clear(Option_t* option = "")
    virtual void Close(Option_t* option = "")
    virtual void Closed()
    virtual void Delete(Option_t* option = "")
```

The class methods list

- There are three types of method: "private", "protected", and "public"

you cannot use the "private" or "protected" methods...

... so we'll only look at the "public" ones

```
private:
    TCanvas(const TCanvas& canvas)
    void Build()
    virtual void CopyPixmaps()
    void DrawEventStatus(Int_t event, Int_t x, Int_t y, TObject* selected)
    TCanvas& operator=(const TCanvas& rhs)
    void RunAutoExec()

protected:
    void Constructor()
    void Constructor(const char* name, const char* title, Int_t form)
    void Constructor(const char* name, const char* title, Int_t ww, Int_t wh)
    void Constructor(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t wright, Int_t wbottom)
    void Destructor()
    virtual void ExecuteEvent(Int_t event, Int_t px, Int_t py)
    void Init()

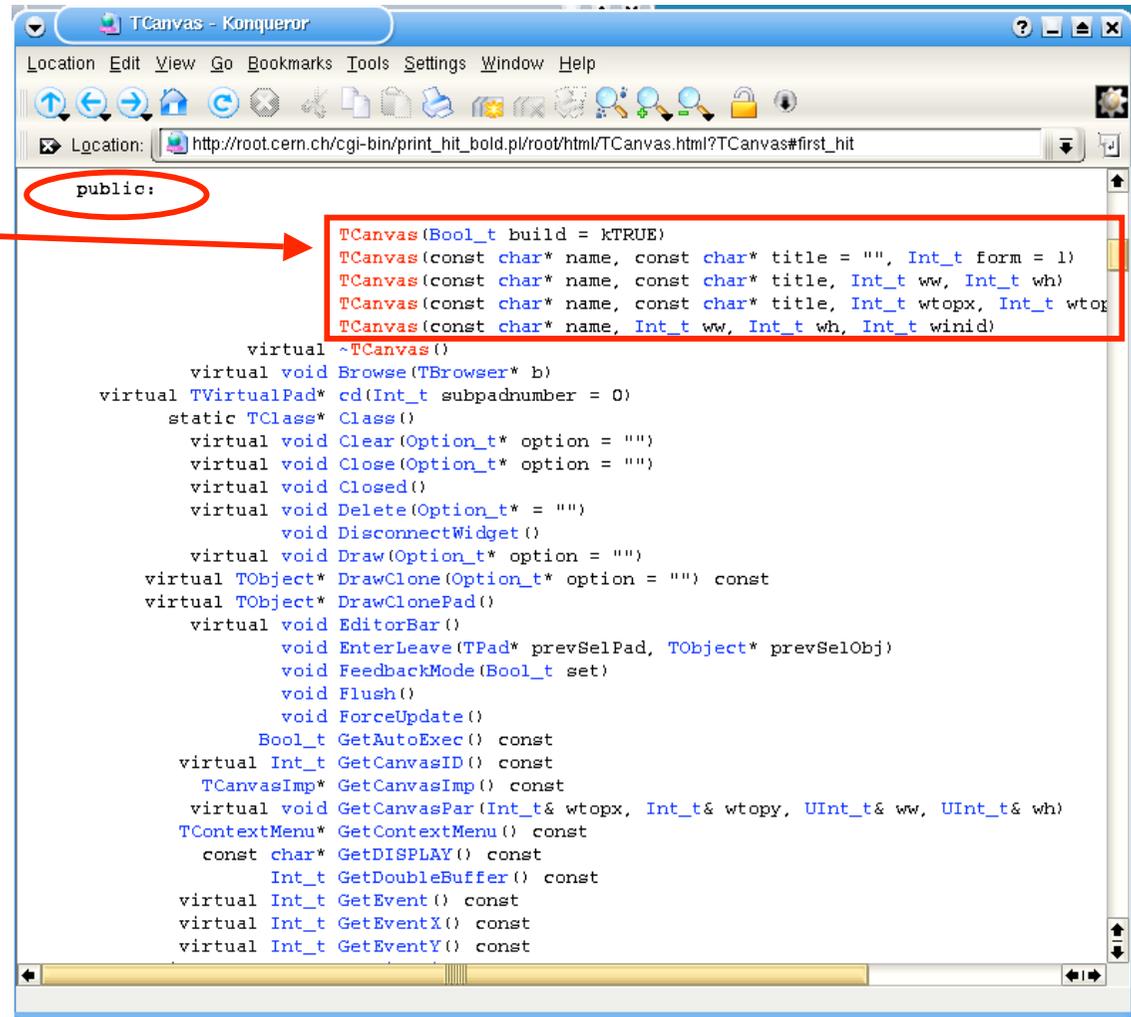
public:
    TCanvas(Bool_t build = kTRUE)
    TCanvas(const char* name, const char* title = "", Int_t form = 1)
    TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
    TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t wright, Int_t wbottom)
    TCanvas(const char* name, Int_t ww, Int_t wh, Int_t winid)

    virtual ~TCanvas()
    virtual void Browse(TBrowser* b)
    virtual TVirtualPad* cd(Int_t subpadnumber = 0)
    static TClass* Class()
    virtual void Clear(Option_t* option = "")
    virtual void Close(Option_t* option = "")
    virtual void Closed()
    virtual void Delete(Option_t* option = "")
```

The class methods list

- The "public" methods list is always organised in the same way:

First the *constructors* of the class (methods with the same name as the class)



```
public:
TCanvas(Bool_t build = kTRUE)
TCanvas(const char* name, const char* title = "", Int_t form = 1)
TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t ww, Int_t wh)
TCanvas(const char* name, Int_t ww, Int_t wh, Int_t wind)

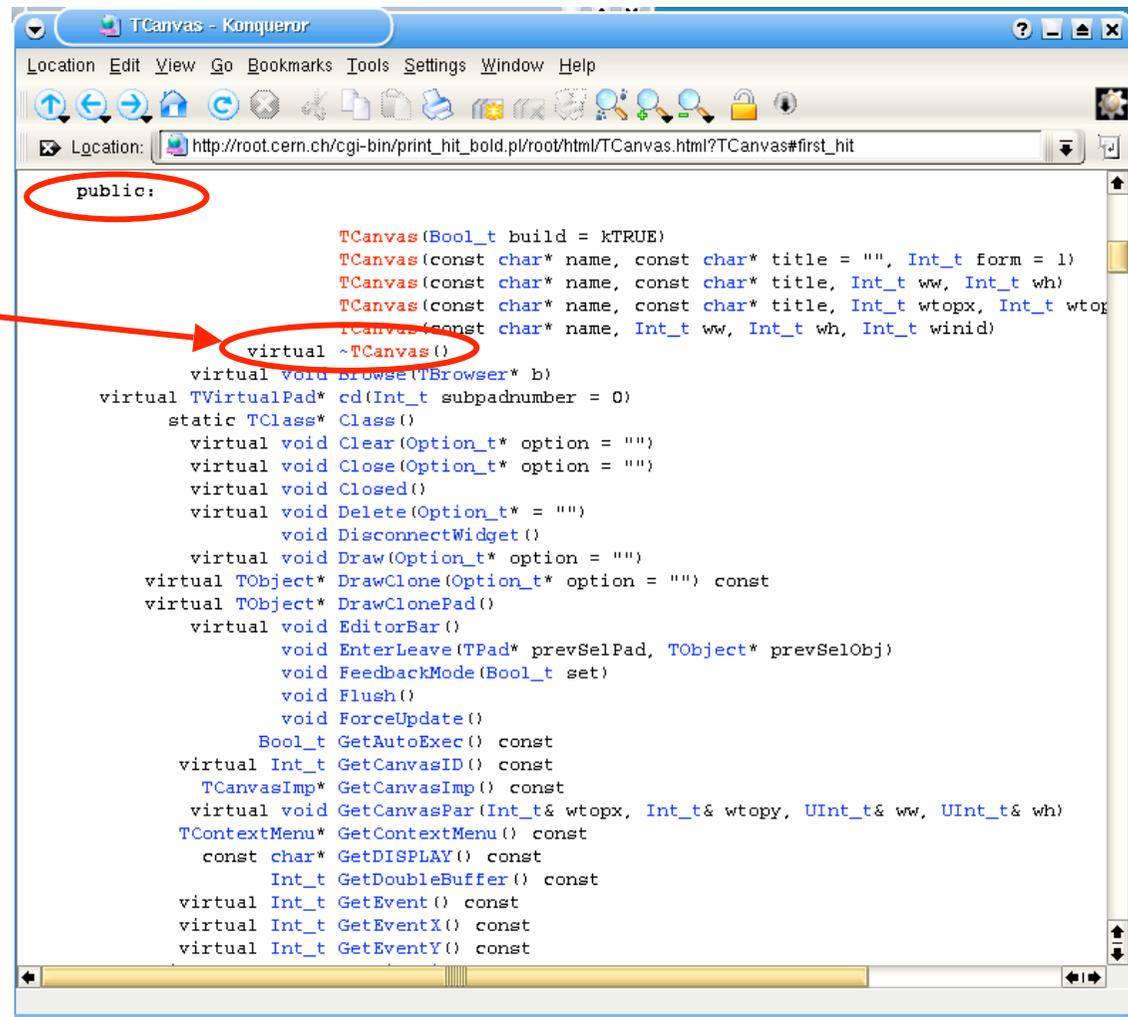
virtual ~TCanvas()
virtual void Browse(TBrowser* b)
virtual TVirtualPad* cd(Int_t subpadnumber = 0)
static TClass* Class()
virtual void Clear(Option_t* option = "")
virtual void Close(Option_t* option = "")
virtual void Closed()
virtual void Delete(Option_t* = "")
void DisconnectWidget()
virtual void Draw(Option_t* option = "")
virtual TObject* DrawClone(Option_t* option = "") const
virtual TObject* DrawClonePad()
virtual void EditorBar()
void EnterLeave(TPad* prevSelPad, TObject* prevSelObj)
void FeedbackMode(Bool_t set)
void Flush()
void ForceUpdate()
Bool_t GetAutoExec() const
virtual Int_t GetCanvasID() const
TCanvasImp* GetCanvasImp() const
virtual void GetCanvasPar(Int_t& wtopx, Int_t& wtopy, UInt_t& ww, UInt_t& wh)
TContextMenu* GetContextMenu() const
const char* GetDISPLAY() const
Int_t GetDoubleBuffer() const
virtual Int_t GetEvent() const
virtual Int_t GetEventX() const
virtual Int_t GetEventY() const
```

The class methods list

- The "public" methods list is always organised in the same way:

A destructor
"`~ClassName()`" called when
the object is destroyed,
e.g. by `delete*`

*the destructor cannot be
called directly i.e.
`toto->~ClassName();`
does not work. we use:
`delete toto;`



```
public:
    TCanvas(Bool_t build = kTRUE)
    TCanvas(const char* name, const char* title = "", Int_t form = 1)
    TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
    TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t wh)
    TCanvas(const char* name, Int_t ww, Int_t wh, Int_t windid)
    virtual ~TCanvas()
    virtual void Browse(TBrowser* b)
    virtual TVirtualPad* cd(Int_t subpadnumber = 0)
    static TClass* Class()
    virtual void Clear(Option_t* option = "")
    virtual void Close(Option_t* option = "")
    virtual void Closed()
    virtual void Delete(Option_t* option = "")
    void DisconnectWidget()
    virtual void Draw(Option_t* option = "")
    virtual TObject* DrawClone(Option_t* option = "") const
    virtual TObject* DrawClonePad()
    virtual void EditorBar()
    void EnterLeave(TPad* prevSelPad, TObject* prevSelObj)
    void FeedbackMode(Bool_t set)
    void Flush()
    void ForceUpdate()
    Bool_t GetAutoExec() const
    virtual Int_t GetCanvasID() const
    TCanvasImp* GetCanvasImp() const
    virtual void GetCanvasPar(Int_t& wtopx, Int_t& wtopy, UInt_t& ww, UInt_t& wh)
    TContextMenu* GetContextMenu() const
    const char* GetDISPLAY() const
    Int_t GetDoubleBuffer() const
    virtual Int_t GetEvent() const
    virtual Int_t GetEventX() const
    virtual Int_t GetEventY() const
```

The class methods list

- The "public" methods list is always organised in the same way:

```
public:
    TCanvas(Bool_t build = kTRUE)
    TCanvas(const char* name, const char* title = "", Int_t form = 1)
    TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
    TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy)
    TCanvas(const char* name, Int_t ww, Int_t wh, Int_t windid)

    virtual ~TCanvas()

    virtual void Browse(TBrowser* b)
    virtual TVirtualPad* cd(Int_t subpadnumber = 0)
    static TClass* Class()
    virtual void Clear(Option_t* option = "")
    virtual void Close(Option_t* option = "")
    virtual void Closed()
    virtual void Delete(Option_t* option = "")
    void DisconnectWidget()
    virtual void Draw(Option_t* option = "")
    virtual TObject* DrawClone(Option_t* option = "") const
    virtual TObject* DrawClonePad()
    virtual void EditorBar()
    void EnterLeave(TPad* prevSelPad, TObject* prevSelObj)
    void FeedbackMode(Bool_t set)
    void Flush()
    void ForceUpdate()
    Bool_t GetAutoExec() const
    virtual Int_t GetCanvasID() const
    TCanvasImp* GetCanvasImp() const
    virtual void GetCanvasPar(Int_t& wtopx, Int_t& wtopy, UInt_t& ww, UInt_t& wh)
    TContextMenu* GetContextMenu() const
    const char* GetDISPLAY() const
    Int_t GetDoubleBuffer() const
    virtual Int_t GetEvent() const
    virtual Int_t GetEventX() const
    virtual Int_t GetEventY() const
```

An alphabetical list of *all* of the methods of the class...?

Where is "Divide" ?

Finding ALL the methods of a class

- If a method seems to be missing from a class, it might be defined by its (grand-)parents...

The objects of a class inherit all the characteristics & know-how of their ancestors...

...whilst adding a few particularities of their own.

The screenshot shows a web browser window titled "TCanvas - Konqueror". The address bar shows the URL: `http://root.cern.ch/cgi-bin/print_hit_bold.pl/root/html/TCanvas.html?TCanvas#first_hit`. The page content includes:

- A header for "TCanvas" with a library reference: `library: libGpad` and `#include "TCanvas.h"`.
- A link for "class description - source file - inheritance tree".
- The class definition: `class TCanvas : public TPad`. The `TPad` is circled in red, with an arrow pointing to it from the text "Start with the parent class".
- An "Inheritance Chart" showing a hierarchy of classes. A red box highlights the following classes: `TObject`, `TAttLine`, `TAttFill <- TVirtualPad`, `TAttPad`, and `TQObject`. Within this box, `TPad` is circled in red, with an arrow pointing to it from the text "Start with the parent class".
- A list of other classes: `TAttFillCanvas`, `TAttLineCanvas`, `TAttMarkerCanvas`, `TAttTextCanvas`, `TDialogCanvas <-`, `TDrawPanelHist`, `TFitPanel <- TFitPanelGraph`, and `TInspectCanvas`.
- Private methods and constructors:

```
private:
    TCanvas(const TCanvas& canvas)
    void Build()
    virtual void CopyPixmaps()
    void DrawEventStatus(Int_t event, Int_t x, Int_t y, TObject* selected)
    TCanvas& operator=(const TCanvas& rhs)
    void DrawAutoExec()
```

Finding ALL the methods of a class

- If a method seems to be missing from a class, it might be defined by its (grand-)parents...

```
public:
    TPad()
    TPad(const char* name, const char* title, Double_t xlow, Double_t xhigh, Double_t ylow, Double_t yhigh)
    virtual ~TPad()
    virtual void AbsCoordinates(Bool_t set)
    virtual Double_t AbsPixeltoX(Int_t px)
    virtual void AbsPixeltoXY(Int_t xpixel, Int_t ypixel, Double_t& x, Double_t& y)
    virtual Double_t AbsPixeltoY(Int_t py)
    virtual void AddExec(const char* name, const char* command)
    virtual void AutoExec()
    virtual void Browse(TBrowser* b)
    virtual TLegend* BuildLegend(Double_t x1 = 0.5, Double_t y1 = 0.67, Double_t x2 = 0.5, Double_t y2 = 0.67)
    virtual TVirtualPad* cd(Int_t subpadnumber = 0)
    static TClass* Class()
    virtual void Clear(Option_t* option = "")
    virtual Int_t Clip(Float_t* x, Float_t* y, Float_t xclip1, Float_t yclip1, Float_t xclip2, Float_t yclip2)
    virtual Int_t Clip(Double_t* x, Double_t* y, Double_t xclip1, Double_t yclip1, Double_t xclip2, Double_t yclip2)
    virtual Int_t ClippingCode(Double_t x, Double_t y, Double_t xcl1, Double_t ycl1, Double_t xcl2, Double_t ycl2)
    virtual void Close(Option_t* option = "")
    virtual void Closed()
    virtual void CloseToolTip(TObject* tip)
    virtual void CopyPixmap()
    virtual void CopyPixmap()
    virtual TObject* CreateToolTip(const TBox* b, const char* text, Long_t delays)
    virtual void DeleteExec(const char* name)
    virtual void DeleteToolTip(TObject* tip)
    virtual void Divide(Int_t nx = 1, Int_t ny = 1, Float_t xmargin = 0.01, Float_t ymargin = 0.01)
    virtual void Draw(Option_t* option = "")
    virtual void DrawClassObject(const TObject* obj, Option_t* option = "")
    static void DrawColorTable()
    virtual void DrawCrosshair()
    virtual TH1F* DrawFrame(Double_t xmin, Double_t ymin, Double_t xmax, Double_t ymax, Double_t xmargin, Double_t ymargin)
    virtual TObject* FindObject(const char* name) const
```

Found it!
Let's look at the explanation...*

*...we'll explain the function declaration (variable types, default arguments etc.) later

Lost & found

Retrieving lost objects

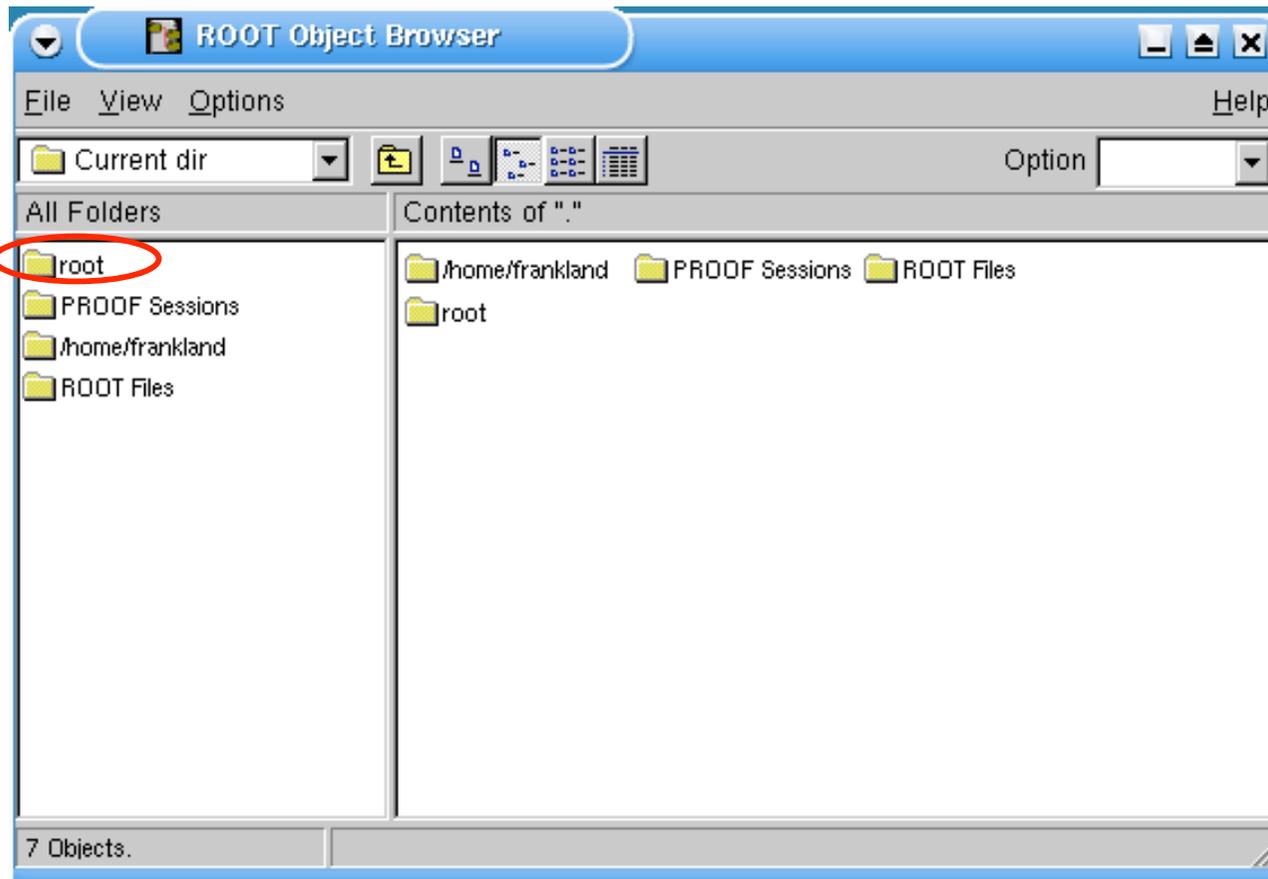
Or:

Why (most) objects have a name

Finding 'lost' objects

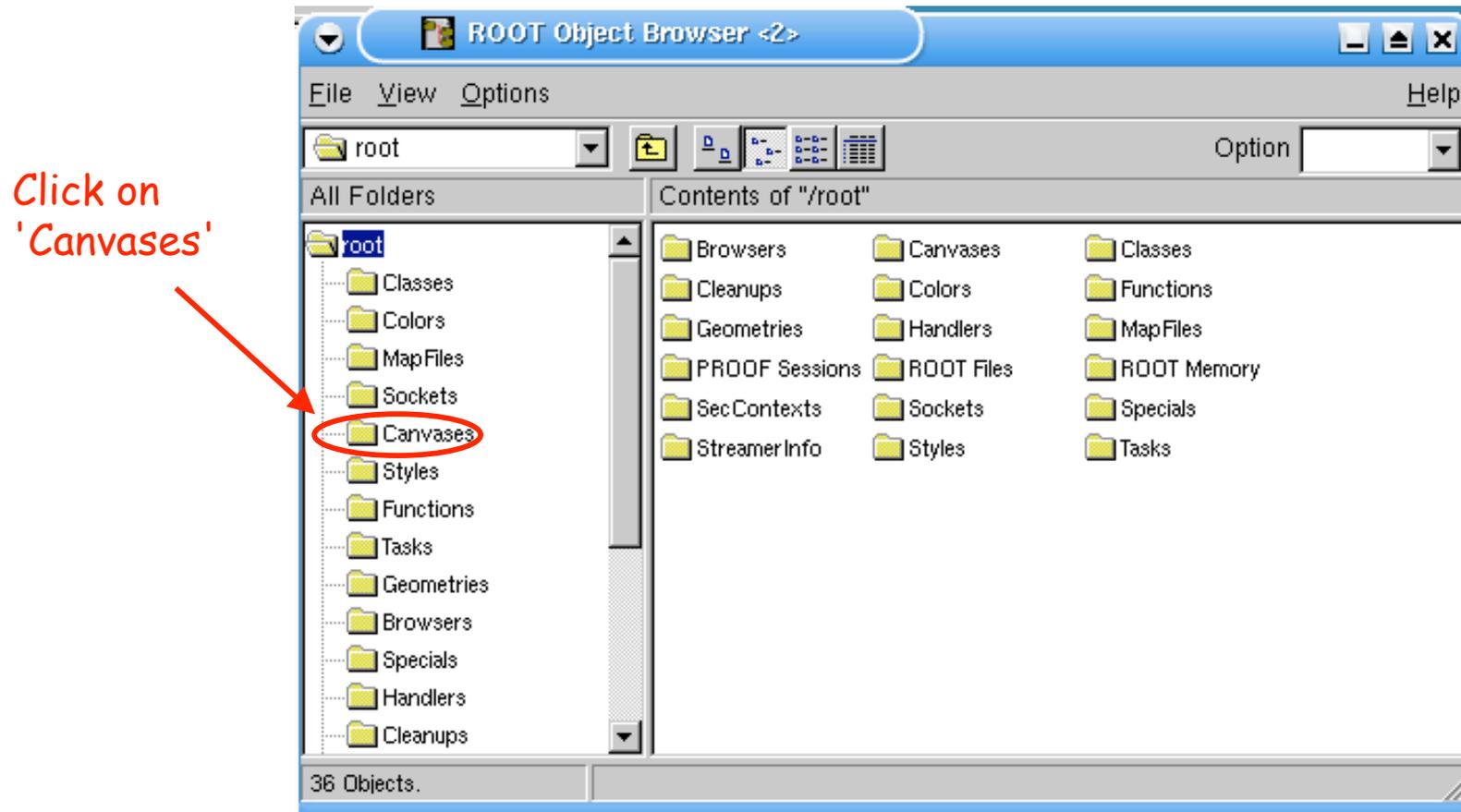
- ROOT keeps lists of objects, allowing to find them easily

Double-click 'root'



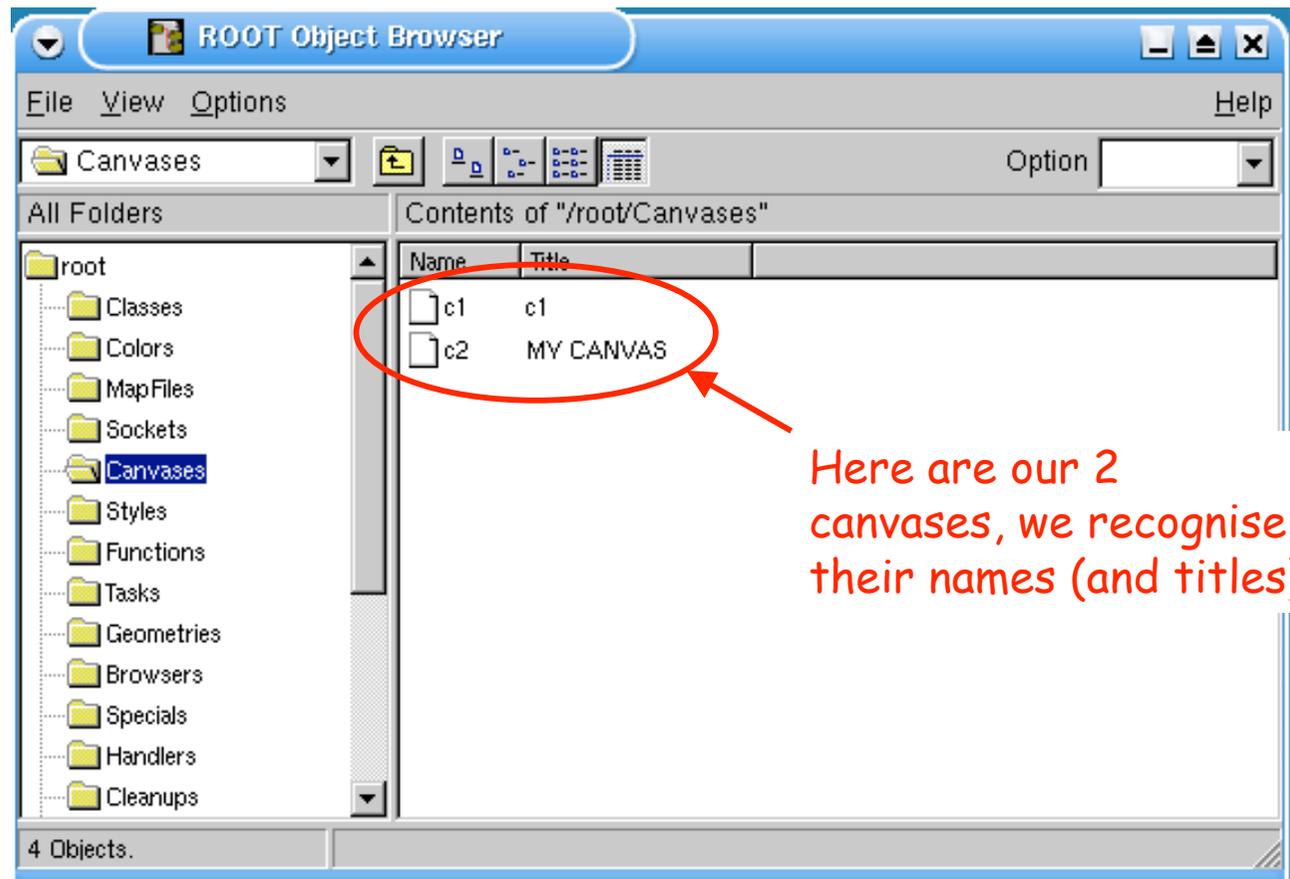
Finding 'lost' objects

- You can browse these lists using the TBrowser



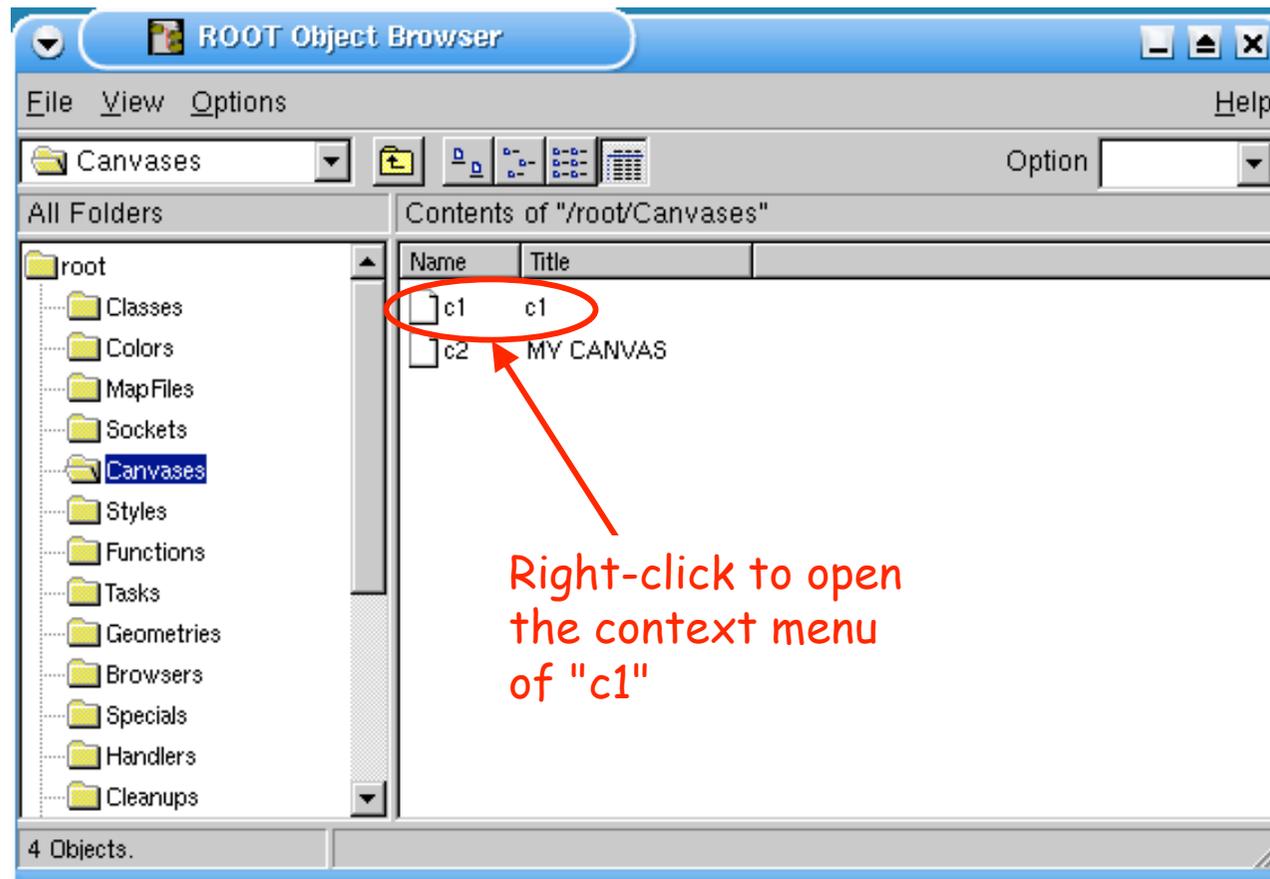
Finding 'lost' objects

- Each object has to have an unique name for us to find it.



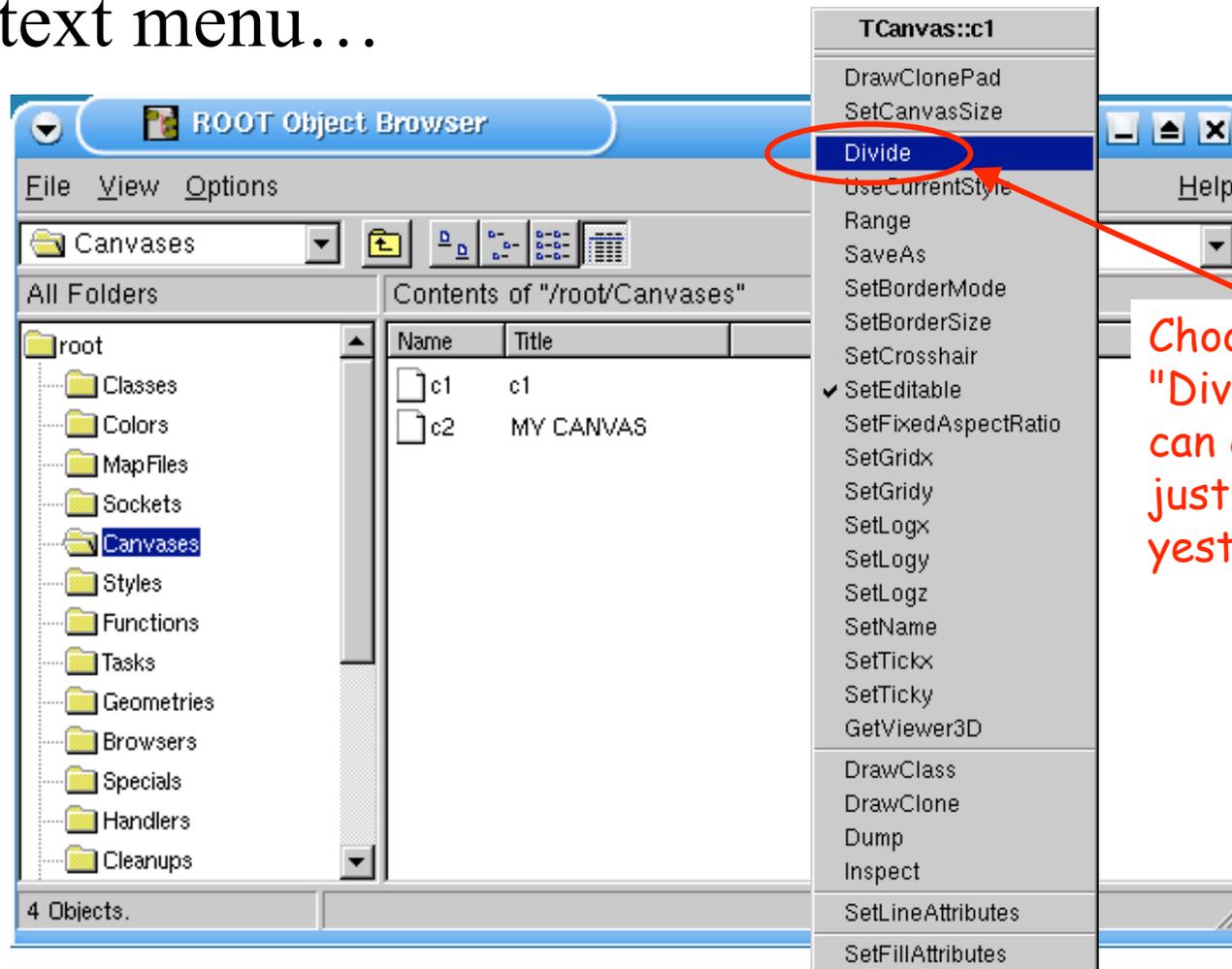
Finding 'lost' objects

- You can interact with the objects using their context menu...



Finding 'lost' objects

- You can interact with the objects using their context menu...



Choose e.g. "Divide", and you can cut "c1" in 4, just like yesterday...

Finding 'lost' objects

- ...or if you knew the address of the object, you could use a pointer:

```
gROOT->GetListOfCanvases()->FindObject("c1");
```

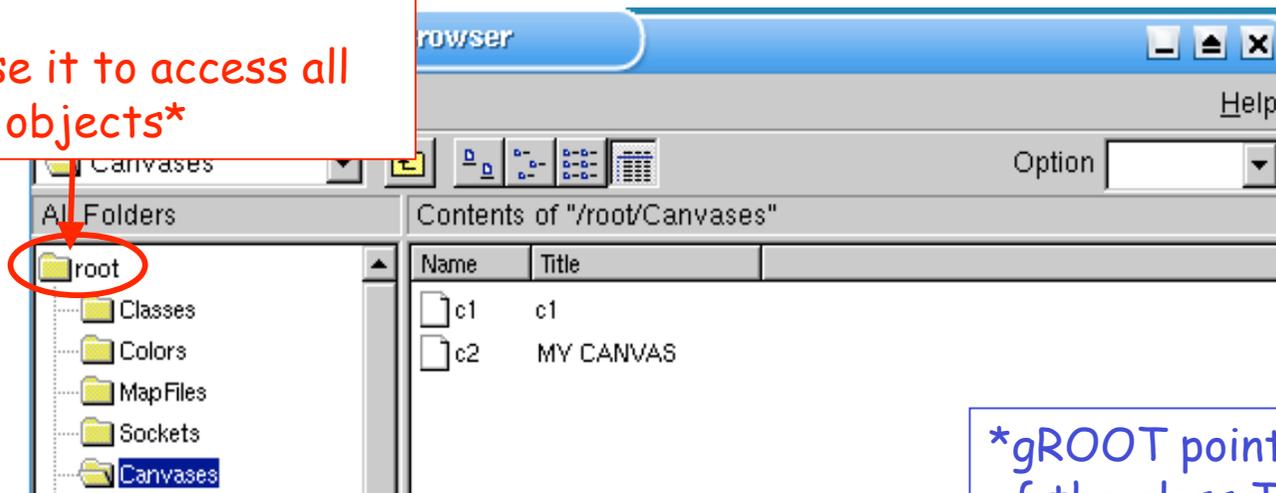
Finding 'lost' objects

- ...or if you knew the address of the object, you could use a pointer:

```
gROOT->GetListOfCanvases()->FindObject("c1");
```

This global pointer contains the address of the 'root' folder of the folder tree.

You can use it to access all the other objects*



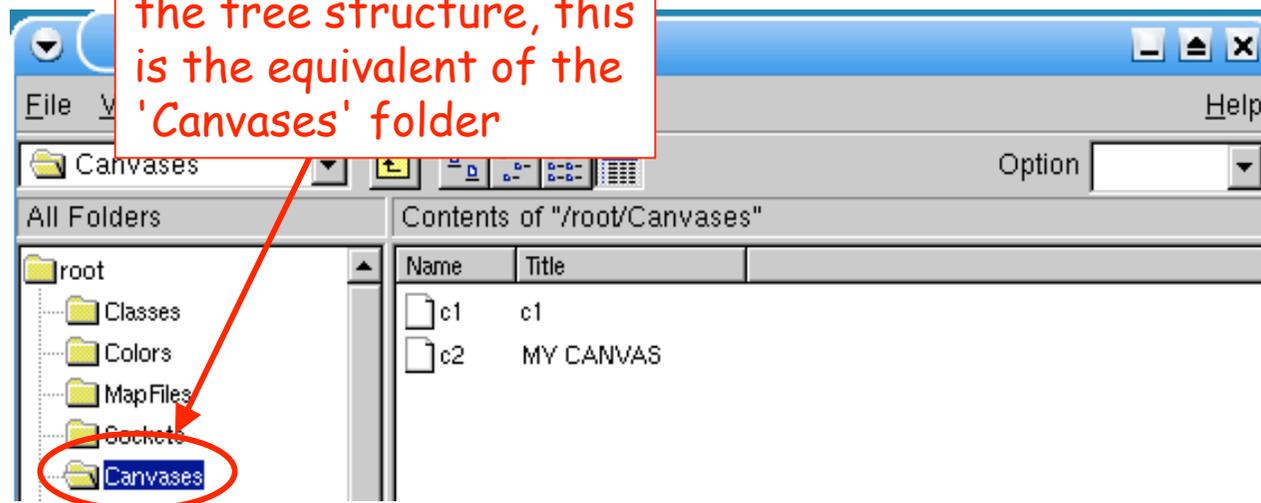
*gROOT points to an object of the class TROOT.

Finding 'lost' objects

- ...or if you knew the address of the object, you could use a pointer:

```
gROOT->GetListOfCanvases()->FindObject("c1");
```

Going down a level in the tree structure, this is the equivalent of the 'Canvases' folder

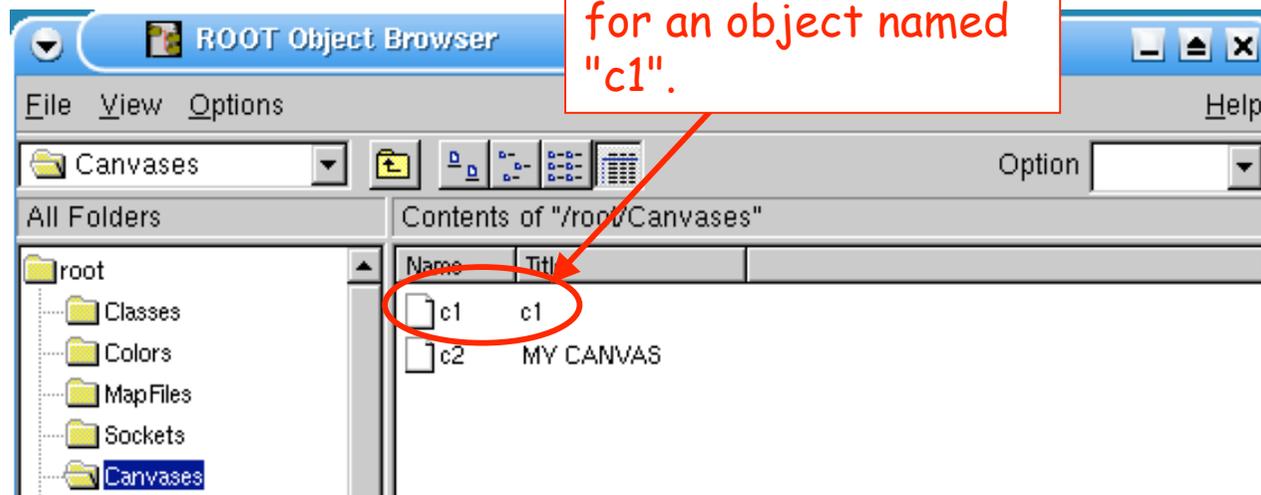


Finding 'lost' objects

- ...or if you knew the address of the object, you could use a pointer:

```
gROOT->GetListOfCanvases()->FindObject("c1");
```

In the list/folder of all Canvases, we look for an object named "c1".



Finding 'lost' objects

- If you aren't sure which folder to look in, you can recursively search through all the folders:

```
gROOT->GetListOfCanvases()->FindObject("c1");
```

```
gROOT->FindObject("name");
```

This is the **MAGIC FORMULA** which allows to find (nearly) any object anywhere anytime.

You'll use it all the time!!

Finding 'lost' objects

- All you have to do now is put the address in the appropriate pointer and use it:

```
ClassName* toto = (ClassName*) gROOT->FindObject("nom");
```



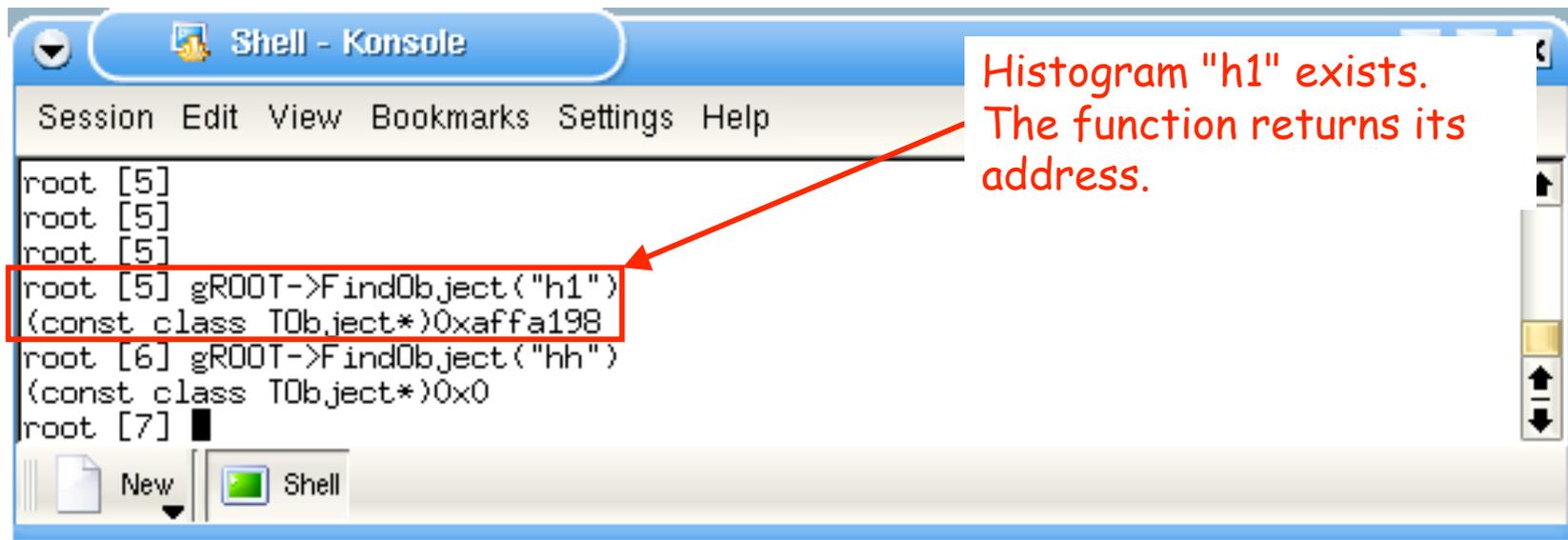
We have to re-specify the class of the object we hope to find here

Example: we look for canvas "c1" and wipe it clean:

```
TCanvas* c1_ptr = (TCanvas*) gROOT->FindObject("c1");  
c1_ptr->Clear();
```

Testing if an object exists

- You use the same function to know if an object with a given name exists or not:



The screenshot shows a terminal window titled "Shell - Konsole" with a menu bar (Session, Edit, View, Bookmarks, Settings, Help). The terminal output is as follows:

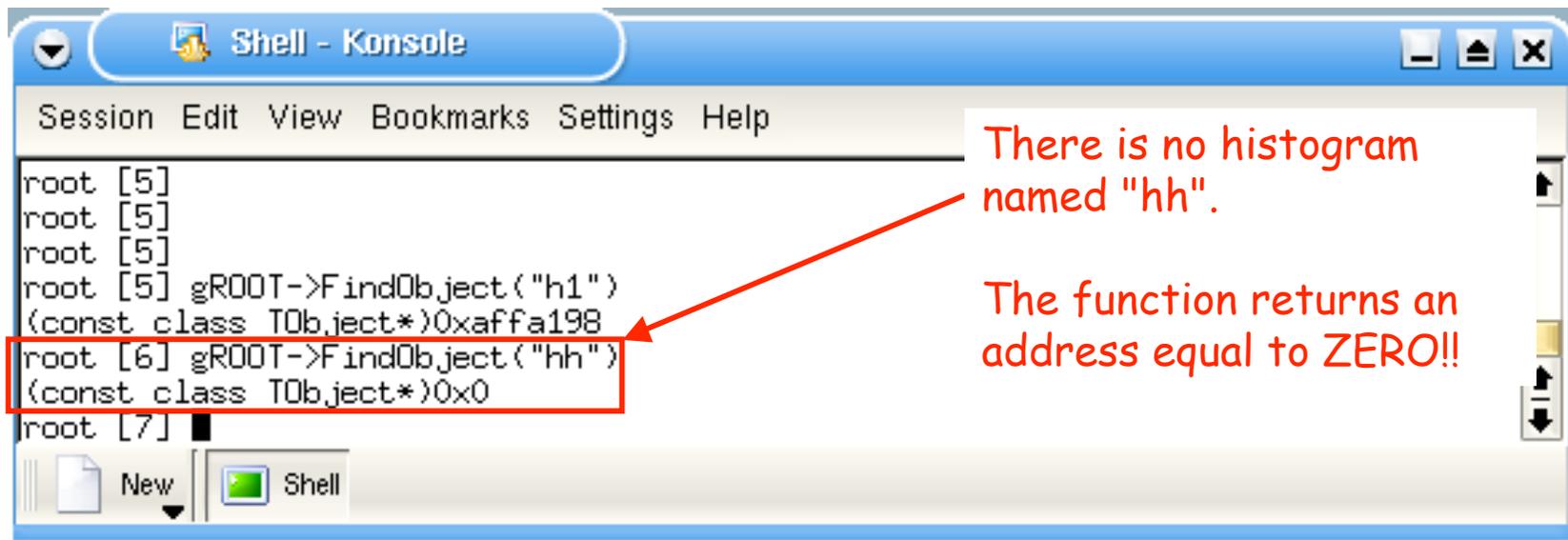
```
root [5]
root [5]
root [5]
root [5] gROOT->FindObject("h1")
(const class TObject*)0xaffa198
root [6] gROOT->FindObject("hh")
(const class TObject*)0x0
root [7] █
```

A red box highlights the output of the first command: `(const class TObject*)0xaffa198`. A red arrow points from a text box to this output.

Histogram "h1" exists.
The function returns its address.

Testing if an object exists

- You use the same function to know if an object with a given name exists or not:



The screenshot shows a ROOT Shell console window titled "Shell - Konsole". The window contains a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The console output shows several lines of text:

```
root [5]
root [5]
root [5]
root [5] gROOT->FindObject("h1")
(const class TObject*)0xaffa198
root [6] gROOT->FindObject("hh")
(const class TObject*)0x0
root [7]
```

The line `root [6] gROOT->FindObject("hh")` and its output `(const class TObject*)0x0` are highlighted with a red box. A red arrow points from a text box on the right to the output of this line.

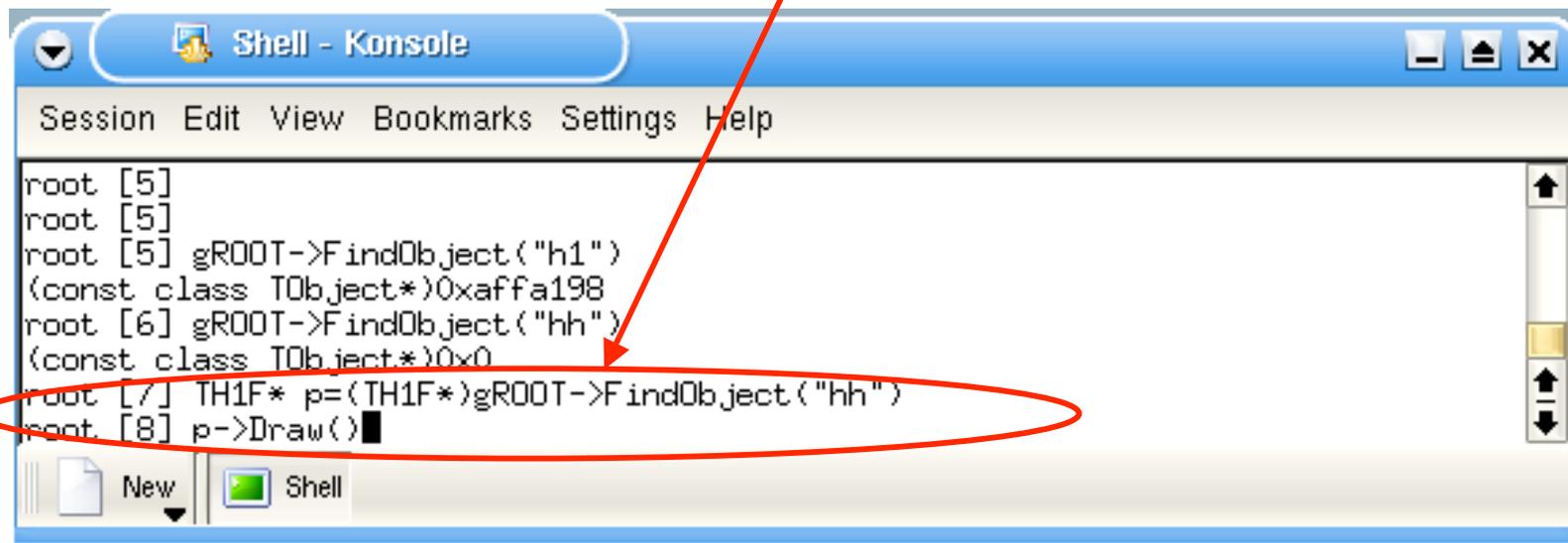
There is no histogram named "hh".

The function returns an address equal to ZERO!!

Testing if an object exists

- To be rigorous, you should always test the value of a pointer before trying to use it...

What happens if you try to use a pointer holding the address 0 ?

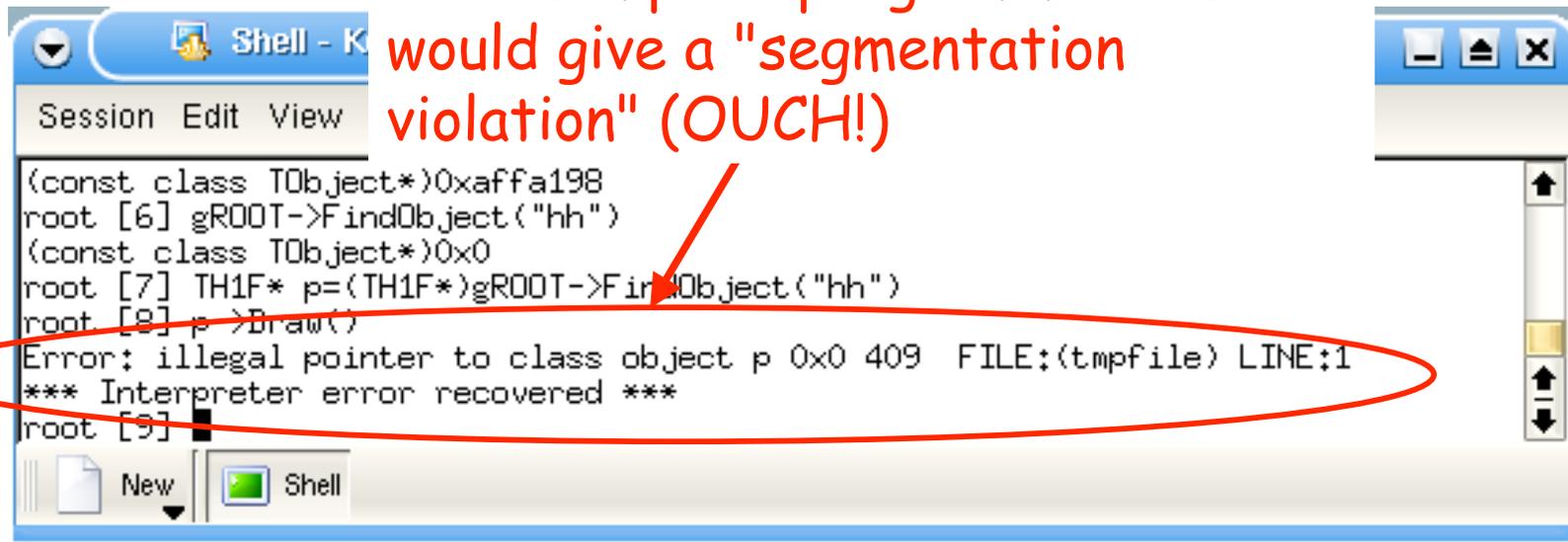


```
Shell - Konsole
Session Edit View Bookmarks Settings Help
root [5]
root [5]
root [5] gROOT->FindObject("h1")
(const class TObject*)0xaffa198
root [6] gROOT->FindObject("hh")
(const class TObject*)0x0
root [7] TH1F* p=(TH1F*)gROOT->FindObject("hh")
root [8] p->Draw()
```

Testing if an object exists

- To be rigorous, you should always test the value of a pointer before trying to use it...

The interpreter is very kind...
...in a compiled programme this
would give a "segmentation
violation" (OUCH!)



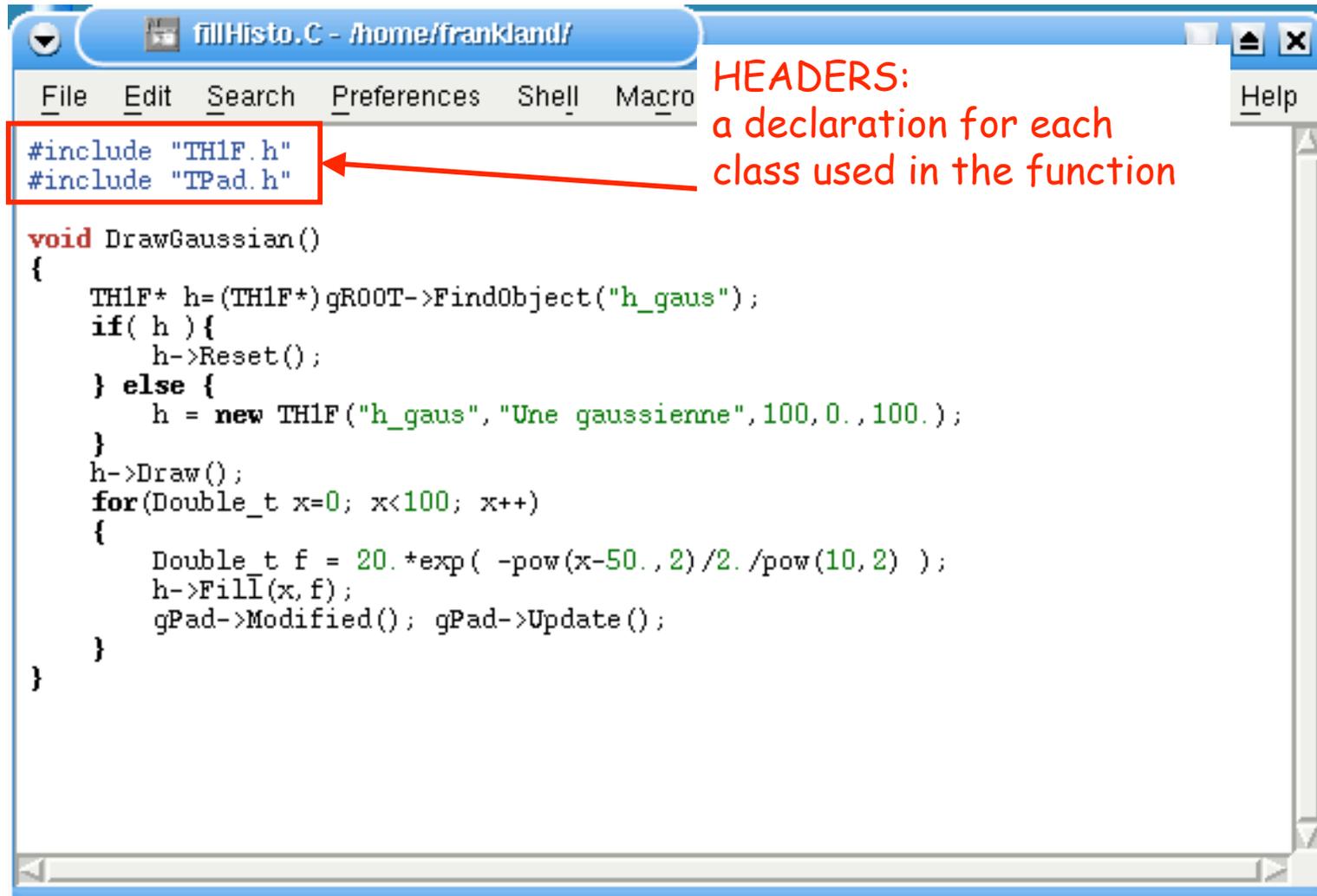
```
Shell - K
Session Edit View
(const class Tobject*)0xaffa198
root [6] gROOT->FindObject("hh")
(const class Tobject*)0x0
root [7] TH1F* p=(TH1F*)gROOT->FindObject("hh")
root [8] p->Draw()
Error: illegal pointer to class object p 0x0 409 FILE:(tmpfile) LINE:1
*** Interpreter error recovered ***
root [9]
```

Function programming

C++ for everybody...

A function

- Here is an example of a C++ function



```
fillHisto.C - /home/frankland/
File Edit Search Preferences Shell Macro Help

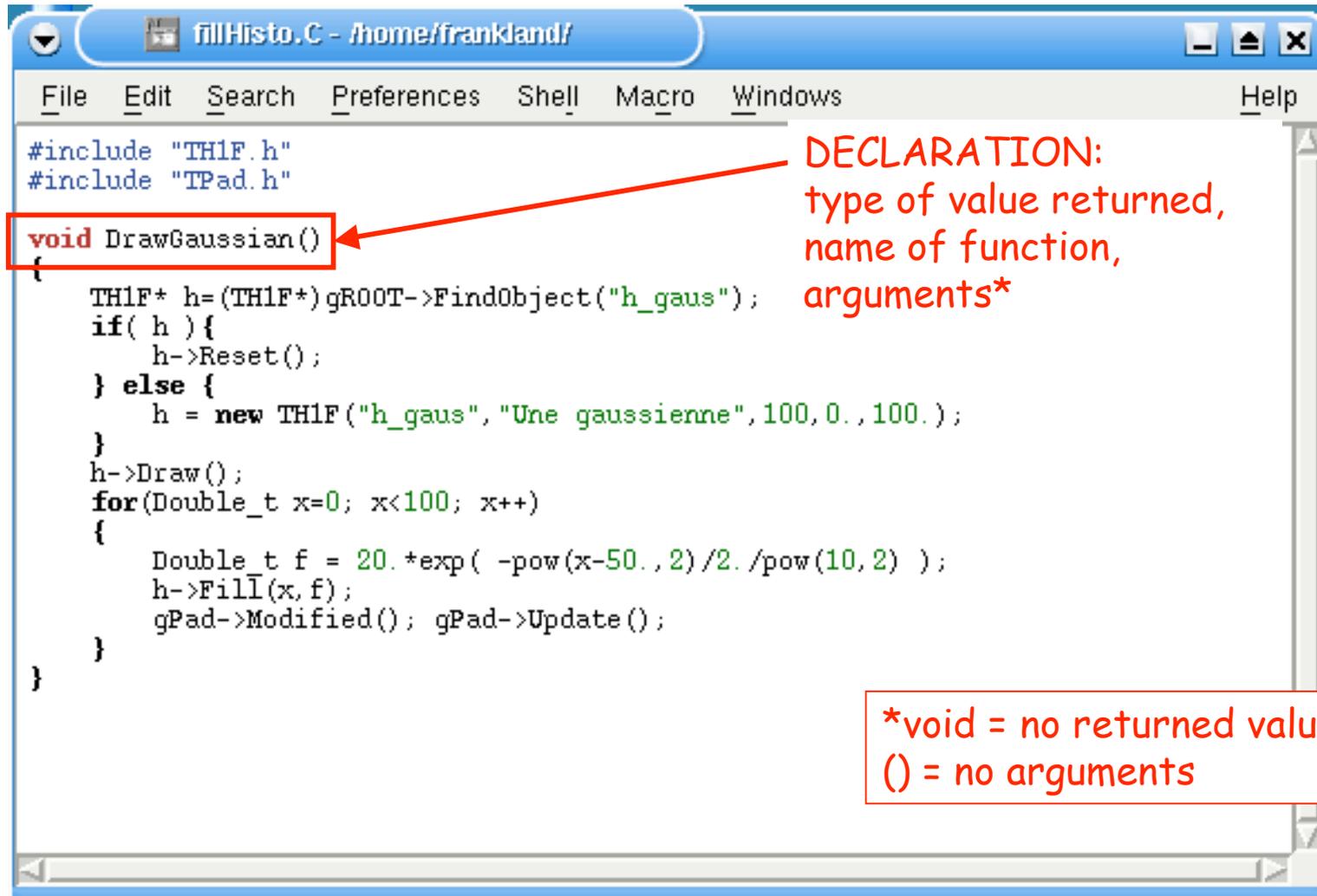
#include "TH1F.h"
#include "TPad.h"

void DrawGaussian()
{
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");
    if( h ){
        h->Reset();
    } else {
        h = new TH1F("h_gaus", "Une gaussienne", 100, 0., 100.);
    }
    h->Draw();
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = 20.*exp( -pow(x-50., 2)/2. /pow(10, 2) );
        h->Fill(x, f);
        gPad->Modified(); gPad->Update();
    }
}
```

HEADERS:
a declaration for each
class used in the function

A function

- Here is an example of a C++ function



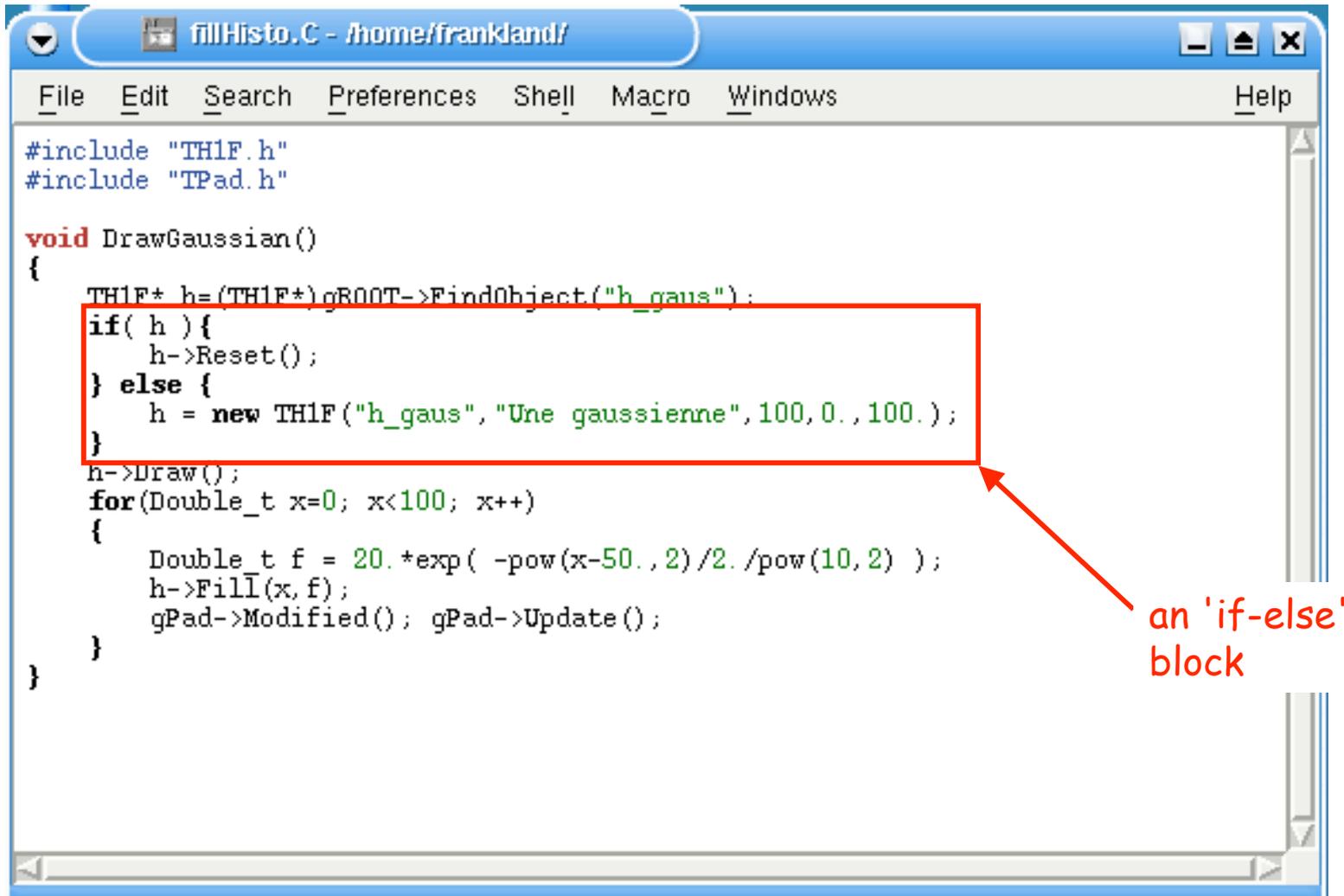
```
fillHisto.C - /home/frankland/  
File Edit Search Preferences Shell Macro Windows Help  
#include "TH1F.h"  
#include "TPad.h"  
void DrawGaussian()  
{  
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");  
    if( h ){  
        h->Reset();  
    } else {  
        h = new TH1F("h_gaus", "Une gaussienne",100,0.,100.);  
    }  
    h->Draw();  
    for(Double_t x=0; x<100; x++)  
    {  
        Double_t f = 20.*exp( -pow(x-50.,2)/2./pow(10,2) );  
        h->Fill(x, f);  
        gPad->Modified(); gPad->Update();  
    }  
}
```

DECLARATION:
type of value returned,
name of function,
arguments*

*void = no returned value
() = no arguments

A function

- Here is an example of a C++ function



```
fillHisto.C - /home/frankland/
File Edit Search Preferences Shell Macro Windows Help

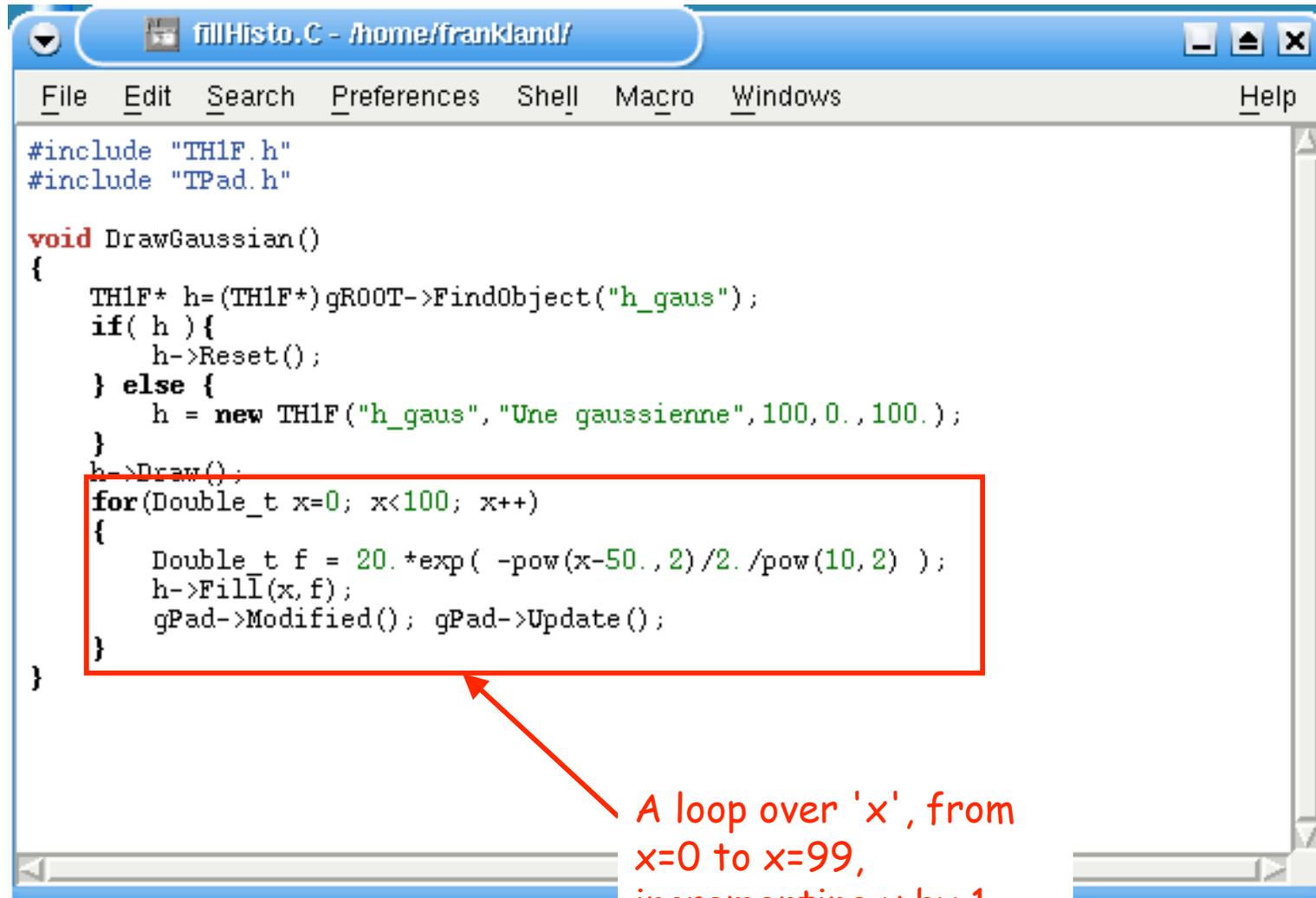
#include "TH1F.h"
#include "TPad.h"

void DrawGaussian()
{
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");
    if( h ){
        h->Reset();
    } else {
        h = new TH1F("h_gaus", "Une gaussienne",100,0.,100.);
    }
    h->Draw();
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = 20.*exp( -pow(x-50.,2)/2./pow(10,2) );
        h->Fill(x, f);
        gPad->Modified(); gPad->Update();
    }
}
```

an 'if-else' block

A function

- Here is an example of a C++ function



```
#include "TH1F.h"
#include "TPad.h"

void DrawGaussian()
{
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");
    if( h ){
        h->Reset();
    } else {
        h = new TH1F("h_gaus", "Une gaussienne",100,0.,100.);
    }
    h->Draw();
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = 20.*exp( -pow(x-50.,2)/2./pow(10,2) );
        h->Fill(x, f);
        gPad->Modified(); gPad->Update();
    }
}
```

A loop over 'x', from x=0 to x=99, incrementing x by 1 each time

A function

- Here is an example of a C++ function

```
fillHisto.C - /home/frankland/
File Edit Search Preferences Shell

#include "TH1F.h"
#include "TPad.h"

void DrawGaussian()
{
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");
    if( h ){
        h->Reset();
    } else {
        h = new TH1F("h_gaus", "Une gaussienne",100,0.,100.);
    }
    h->Draw();
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = 20.*exp( -pow(x-50.,2)/2./pow(10,2) );
        h->Fill(x,f);
        gPad->Modified(); gPad->Update();
    }
}
```

Clear out the contents of the spectrum

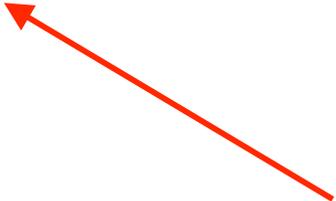
gPad = pointer to the currently active pad or canvas

Compiling and using the function

- To compile and load the definition of the function:

`.L fillHisto.C+`

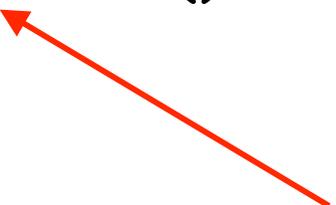
Name of file
containing the
function definition



- To execute the function:

`DrawGaussian()`

Name of the function
(don't forget the '()' !)



Why use 'new' to create the histogram ?

- Let's see what happens if we don't use "new"

The screenshot shows a code editor window titled "fillHistoLocal.C - /home/frankland/". The code defines a function `DrawGaussian0()` that creates a `TH1F` histogram object `h` and fills it with a Gaussian distribution. The code is as follows:

```
#include "TH1F.h"
#include "TPad.h"

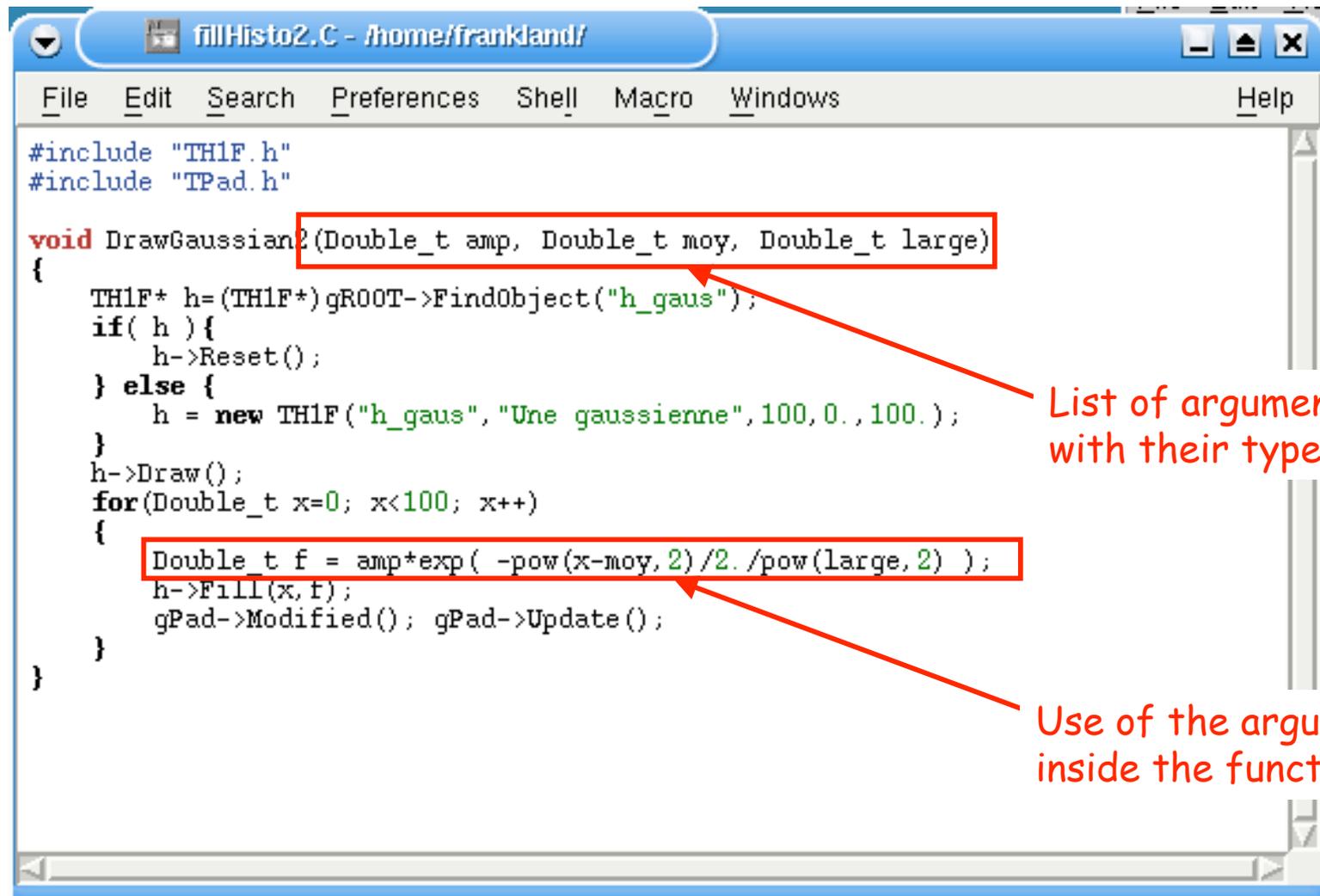
void DrawGaussian0()
{
    TH1F h("h_gaus", "Une gaussienne", 100, 0., 100.);
    h.Draw();
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = 20.*exp(-pow(x-50., 2)/2./pow(10, 2) );
        h.Fill(x, f);
        gPad->Modified(); gPad->Update();
    }
}
```

Three red arrows point to specific parts of the code with the following annotations:

- An arrow points to the line `TH1F h("h_gaus", "Une gaussienne", 100, 0., 100.);` with the text: "Temporary object, it only exists inside this block (function)".
- An arrow points to the `h.Fill(x, f);` line inside the loop with the text: "We see the histogram filling up...".
- An arrow points to the closing brace `}` of the function with the text: "... and then disappear at the end of the function".

Using functions with arguments

- A function with arguments:



```
fillHisto2.C - /home/frankland/
File Edit Search Preferences Shell Macro Windows Help

#include "TH1F.h"
#include "TPad.h"

void DrawGaussian2(Double_t amp, Double_t moy, Double_t large)
{
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");
    if( h ){
        h->Reset();
    } else {
        h = new TH1F("h_gaus", "Une gaussienne",100,0.,100.);
    }
    h->Draw();
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = amp*exp( -pow(x-moy,2)/2./pow(large,2) );
        h->Fill(x,f);
        gPad->Modified(); gPad->Update();
    }
}
```

List of arguments with their type

Use of the arguments inside the function

Arguments with default values

```
fillHisto3.C - /home/frankland/
File Edit Search Preferences Shell Macro Windows Help
#include "TH1F.h"
#include "TPad.h"

void DrawGaussian3(Double_t amp = 20., Double_t moy = 50., Double_t large = 10.)
{
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");
    if( h ){
        h->Reset();
    } else {
        h = new TH1F("h_gaus", "Une gaussienne",100,0.,100.);
    }
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = amp*exp( -pow(x-moy, 2) /2. /pow(large, 2) );
        h->Fill(x, f);
    }
    h->Draw(); gPad->Modified(); gPad->Update();
}
```

e.g. default width for the gaussian

<code>DrawGaussian()</code>	\Rightarrow amp=20, moy=50, large=10
<code>DrawGaussian(5)</code>	\Rightarrow amp=5, moy=50, large=10
<code>DrawGaussian(5,30)</code>	\Rightarrow amp=5, moy=30, large=10
<code>DrawGaussian(5,30,5)</code>	\Rightarrow amp=5, moy=30, large=5

Returning values/objects

- Functions can return ALL sorts of variables...

```
fillHisto4.C - /home/frank/...
File Edit Search Preferences Help

#include "TH1F.h"
#include "TPad.h"

Double_t Gaussian(Double_t z, Double_t toto, Double_t tata, Double_t tutu)
{
    Double_t f = toto*exp( -pow(z-tata,2)/2./pow(tutu,2) );
    return f;
}

TH1F* DrawGaussian4(Double_t amp = 20., Double_t moy = 50., Double_t large = 10.)
{
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");
    if( h ){
        h->Reset();
    } else {
        h = new TH1F("h_gaus", "Une gaussienne", 100,
    }
    for(Double_t x=0; x<100; x++)
    {
        Double_t f = Gaussian(x, amp, moy, large);
        h->Fill(x, f);
    }
    return h;
}
```

Type of returned value

We return the value of 'f'

Gaussian function

Returning values/objects

- Functions can return ALL sorts of variables...

```
fillHisto4.C - /home/frankland/  
File Edit Search Preferences Shell Macro Windows Help  
#include "TH1F.h"  
#include "TROOT.h"  
Double_t G: toto, Double_t tata, Double_t tutu)  
{  
    Double_t f = toto*exp( -pow(z-tata,2)/2./pow(tutu,2) );  
    return f;  
}  
TH1F* DrawGaussian4(Double_t amp = 20., Double_t moy = 50., Double_t large = 10.)  
{  
    TH1F* h=(TH1F*)gROOT->FindObject("h_gaus");  
    if( h ){  
        h->Reset();  
    } else {  
        h = new TH1F("h_gaus", "Une gaussienne", 100, 0., 100.);  
    }  
    for(Double_t x=0; x<100; x++)  
    {  
        Double_t f = Gaussian(x, amp, moy, large);  
        h->Fill(x, f);  
    }  
    return h;  
}
```

We return a pointer to a histogram object

Here we use the gaussian function

We return the value of 'h' i.e. the address of the histogram

Returning values/objects

- Compiling and using the functions:

We can use the gaussian function independently of DrawGaussian4()

```
Shell - Konsole
Session Edit View Bookmarks Settings Help
root [0] ./L fillHisto4.C+
Info in <TUnixSystem::ACLIC>: creating shared library /home/frankland/./fillHist
o4_C.dll
root [1] Gaussian(4,3,2,1)
(Double_t)4.06005849709838107e-01
root [2] IH1F* qq=DrawGaussian4()
root [3] qq->Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [4] █
```

Execute the function DrawGaussian4(), stock the address of the histogram in a pointer and plot it...

Analysis example



Analysis example

An example of an analysis script: we read the data from an ASCII file `basic.dat`; generate a few histograms; and save them in the file `basic.root`.

<http://caeinfo.in2p3.fr/root/Formation/en/Day2/basic.dat>

```
basic.C - /home/frank
File Edit Search Preferences WS

#include "Riostream.h"
#include "TFile.h"
#include "TH1.h"
#include "TNTuple.h"

void basic(const Char_t* fdata="basic.dat", const Char_t* froot="basic.root")
{
    // example of macro to read data from an ascii file and
    // create a root file with an histogram and an ntuple.
    // Arguments :
    // fdata = nom du fichier ascii a lire (default : basic.dat)
    // froot = nom du fichier ROOT resultat (default : basic.root)

    ifstream in;

    // we assume a file basic.dat in the current directory
    // this file has 3 columns of float data

    in.open(fdata, ios::in);
}
```

Declaration of the I/O system of ROOT/C++

this variable type means 'a string'

Object for reading an ASCII file

opening the ASCII file

Analysis example

```
Float_t x,y,z;  
Int_t nlines = 0;  
  
TFile *f = new TFile(froot, "RECREATE");  
  
TH1F *h1 = new TH1F("h1", "x distribution", 100, -4, 4);  
TNTuple *ntuple = new TNTuple("ntuple", "data from ascii file", "x:y:z");
```

Declaring a few working variables

Creation of a new ROOT file, if it exists already we overwrite (recreate) it

Creation of a Ntuple

All histograms, Ntuples, trees (Day 4) created *after* the creation of the file belong to the file...

Analysis example

A 'while' loop: keeps executing as long as the condition is true i.e. as long as the file is readable

```
basic.C - /home/frankl...
File Edit Search Preferences Shell Ma Help
while (in.good())
{
  in >> x >> y >> z;
  if (in.good())
  {
    if (nlines < 5)
    {
      cout << "X = " << x << ", Y = " << y ;
      cout << ", Z = " << z << endl;
    }
    hl->Fill(x);
    ntuple->Fill(x,y,z);
    nlines++;
  }
}
```

we read three Double_t values from the file

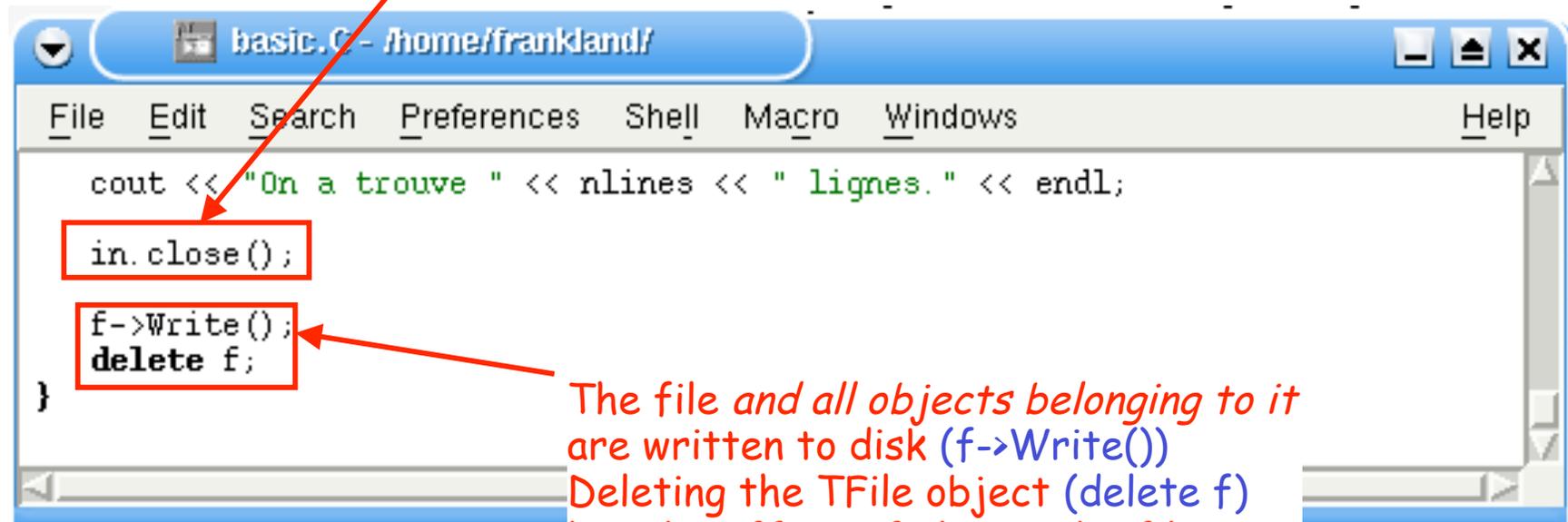
Print the values we read on the screen
endl => carriage-return

Fill the histogram and the ntuple with the values we read

Increment the number of lines read

Analysis example

Close the ASCII file



```
cout << "On a trouve " << nlines << " lignes." << endl;
in.close();
f->Write();
delete f;
}
```

The file and all objects belonging to it are written to disk (f->Write())
Deleting the TFile object (delete f) has the effect of closing the file.

WARNING:

once the file has been closed, the objects belonging to it no longer exist in memory (only on disk)!

Perform the analysis and look at the results

- Compile and execute:

```
.L basic.C+  
basic()
```

- Open the file and plot spectrum 'h1':

```
TFile* fich = new TFile("basic.root")  
fich->ls()  
TH1F* histo = (TH1F*)fich->Get("h1")  
histo->Draw()
```

Print the
contents of
the file



The last two lines in one:
fich->Get("h1")->Draw()

Copy the object 'h1'
into the memory.
Return the address of
this copy.

Closing remarks

Some hints and tips for those dark moments of the soul (when it's just you and the C++ compiler)

Examples of functions/scripts

- On the ROOT web site, under the heading [Tutorials](#), you can find lots of useful examples
- Caution! if the file doesn't have a proper function declaration then you can't compile it:

Executing a
script without
full declaration `.x toto.C`

*In this case, the code will be
interpreted, not compiled.*

WARNING: we really don't
recommend you do this for
your own scripts!!

The 'rootlogon.C' file

- This script without a function declaration is executed automatically when ROOT is launched from the same directory as the file

```
{  
gStyle->SetPalette(1);  
cout << "Salut " << gSystem->Getenv("USER") << "!" << endl;  
gSystem->Exec("date");  
}
```

This way all your 2-D histograms will have nice colours (aah!)

Returns the value of the system environment variable 'USER'

Executes the system command 'date'

For more information, see classes TStyle & TSystem

Exercise

Another example of analysing data

Exercise

(Episode 1: The ROOT menace)

http://caeinfo.in2p3.fr/root/Formation/en/Day2/exo_j2.data

- Write a script to analyse the data in ASCII file **exo_j2.data** . Each line of the file has 4 values corresponding to variables x,y,z et e (How original!)

**x de -25 à 25, y de -25 à 25,
z de -10 à 10, e de -500 à 2500**

- Create and fill the following histograms:
 - 1-D: 'z' distribution (**TH1F**)
 - 2-D: 'y' vs 'x', 'z' vs 'x', 'z' vs 'y' (**TH2F**)
 - Profiles: <e> vs x, <e> vs y (**TProfile**)
 - 3-D: 'z' vs 'y' vs 'x', 'e' vs 'y' vs 'x' (**TH3F**)
- Save them in a ROOT file called **exo_j2.root** .

Exercise

(Episode 2: The return of *exo_j2.root*)

- Open **exo_j2.root**.
- Find the 3 most populated intervals of 'z'
 - *Keep a note of them!*
- Using the FitPanel:
 - fit the TProfile "<e> vs x" with a polynomial and *note* the values of the last two parameters, *ex1* et *ex2*.
 - fit the TProfile "<e> vs y" with a polynomial and *note* the values of the last 3 parameters, *ey1*, *ey2* et *ey3*.
- Close **exo_j2.root**.

Exercise

(Part 3: The analysis strikes back)

- Write another script to analyse **exo_j2.data**.
 - Create and fill the following:
 - 1-D: histogram (**TH1F**) of $de=e-ex1*x-ex2*x*x-ey1*y-ey2*y*y-ey3*y*y*y$
 - 2-D: 'y' vs 'x' for each 'z' interval determined in Part 2 (**TH2F**)
 - 2-D Profiles: $\langle z \rangle$ vs y vs x, $\langle e \rangle$ vs y vs x (**TProfile2D**)
 - ADD them to file **exo_j2.root**.

Exercise

(Part 4: The final shot)

- Find the width of the **de** distribution by fitting with a gaussian.
- Write a script to display, on the same canvas, the following 4 figures:
 - 'y' vs 'x' for $z < 1$ with option **col**
 - 'y' vs 'x' for $3 < z < 5$ with option **box**
 - $\langle z \rangle$ vs y vs x with option **zcol**
 - $\langle e \rangle$ vs y vs x with option **surf1**
- Save the picture in a ".gif" file