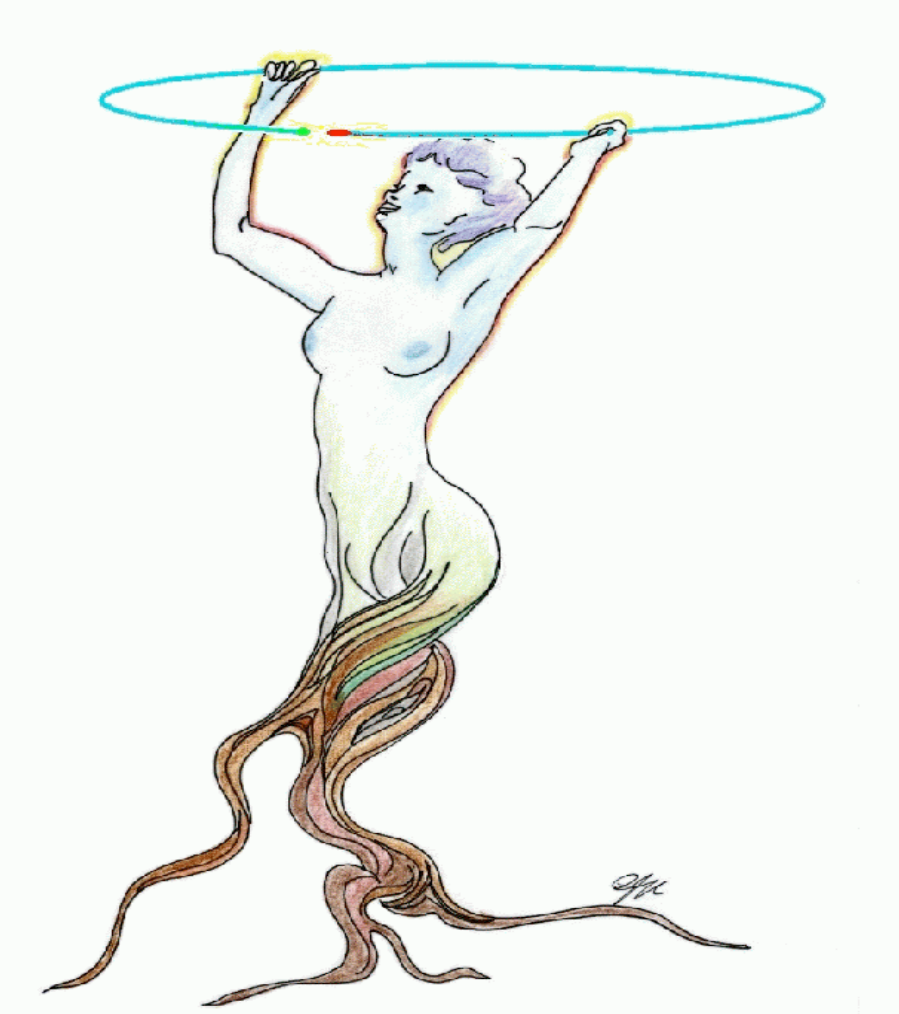
Formation ROOT pour débutants

Deuxième Jour

Programmation



Création et destruction d'objets

Les commandes "new" et "delete"

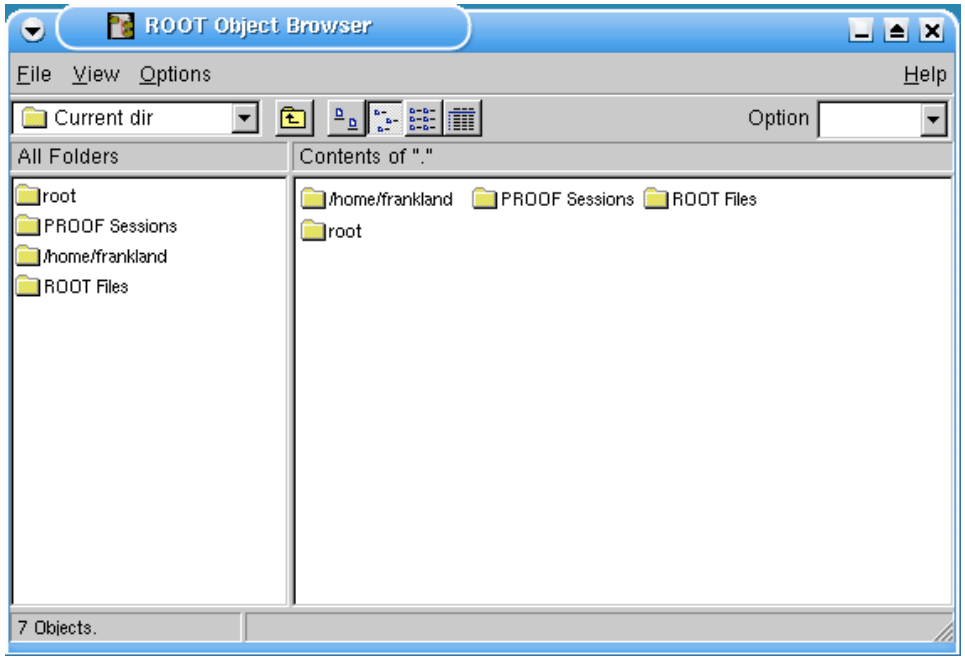
Les pointeurs d'objet

La commande "new"

- On a commencé la journée d'hier en tapant:

```
new TBrowser
```

ce qui a fait apparaître le "ROOT object browser".



La commande "new"

- On a commencé la journée d'hier en tapant:

```
new TBrowser
```

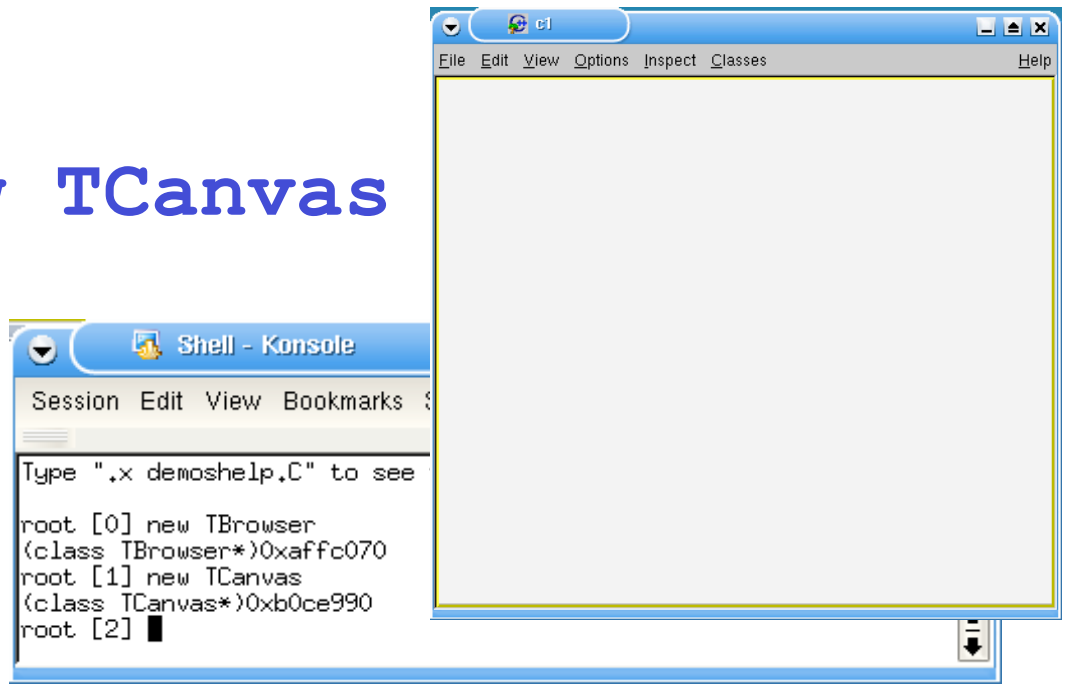
ce qui a fait apparaître le "ROOT object browser".

- Si l'on tape

```
new TCanvas
```

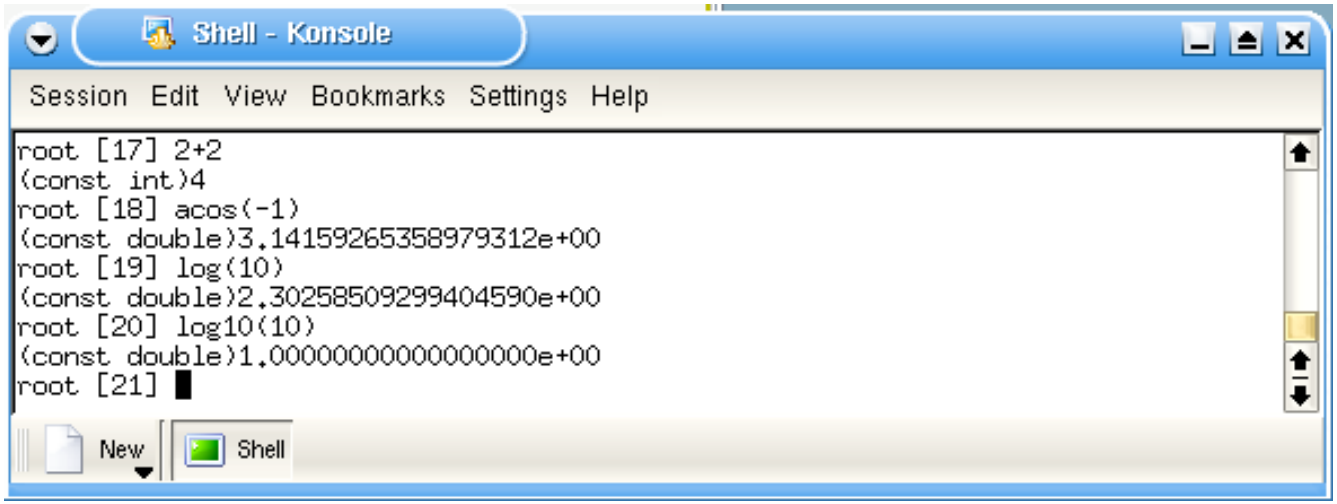
un canevas apparaît!

Mais qu'est-ce qui se passe dans la fenêtre de commandes ???



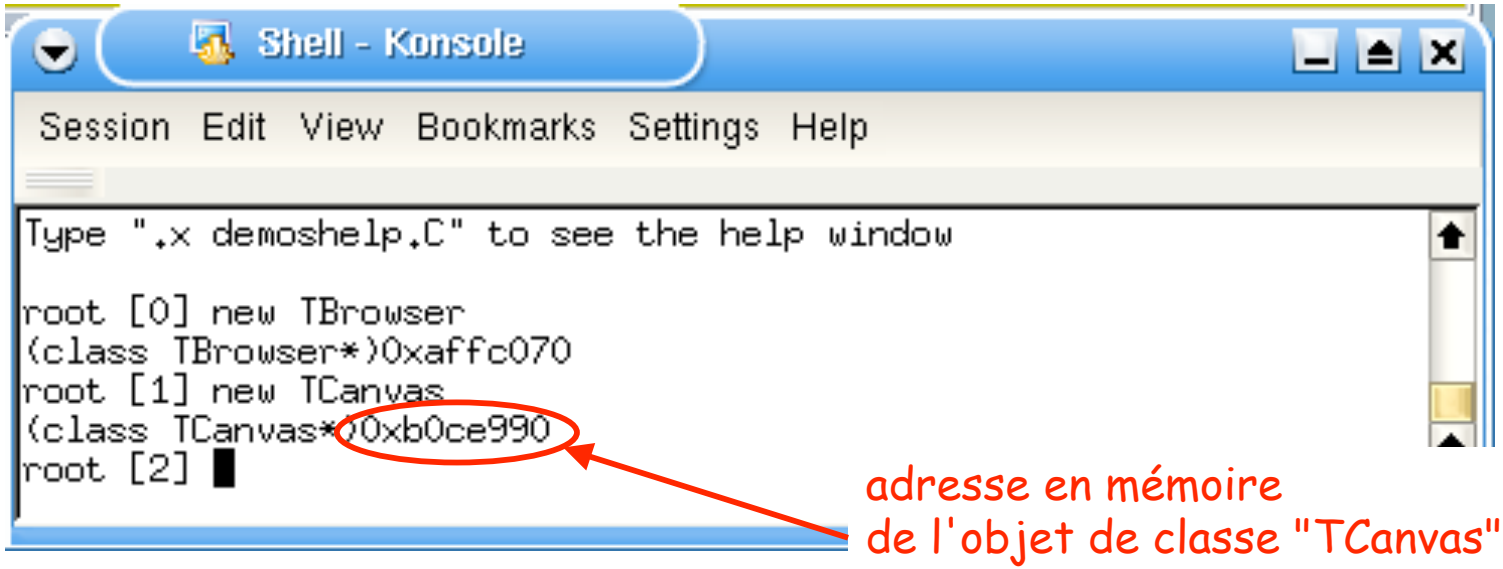
La fenêtre de commandes

- La fenêtre de commandes est un interpréteur de C++ !!
- L'interpréteur affiche la valeur de chaque fonction, expression ou commande que l'on tape - essayez e.g. 2+2…



La fenêtre de commandes

- Les valeurs affichées après une commande **`new`** sont le type (*classe*) et l'adresse en mémoire des objets créés



Les pointeurs d'objet

- Pour utiliser l'objet, il faut mettre son adresse dans une variable spéciale, un *pointeur d'objet*

Déclaration d'un pointeur d'objet :

```
TypeObjet* toto;
```

Les pointeurs d'objet

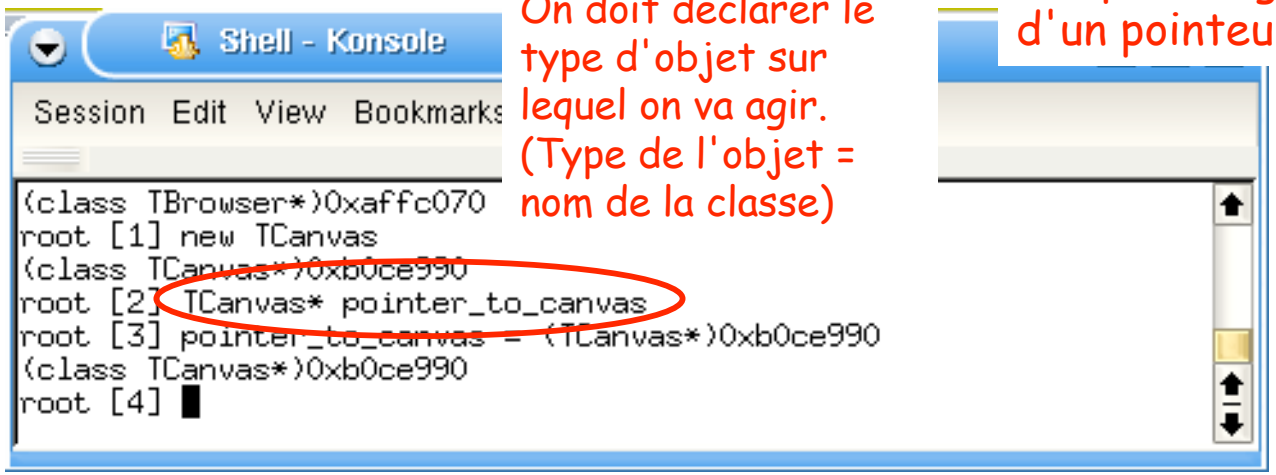
- Pour utiliser l'objet, il faut mettre son adresse dans une variable spéciale, un *pointeur d'objet*

Déclaration d'un pointeur d'objet :

TypeObjet* toto;

On doit déclarer le type d'objet sur lequel on va agir. (Type de l'objet = nom de la classe)

C'est '*' qui nous dit qu'il s'agit d'un pointeur



Les pointeurs d'objet

- La valeur stockée par le pointeur est réellement l'adresse de l'objet en mémoire

Initialisation d'un pointeur d'objet :

```
toto = (TypeObjet*)adresse ;
```

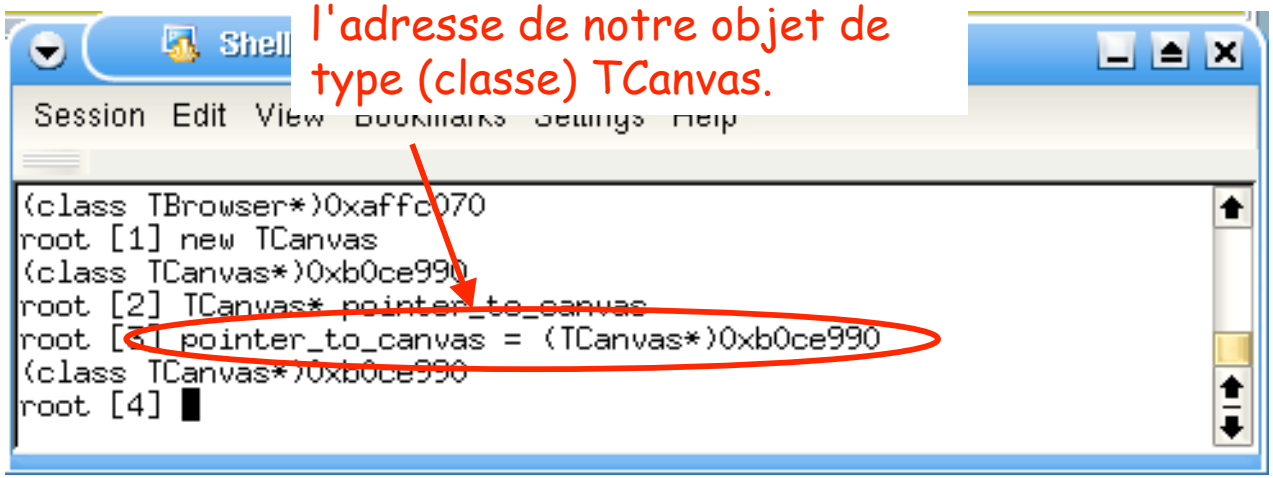
Les pointeurs d'objet

- La valeur stockée par le pointeur est réellement l'adresse de l'objet en mémoire

Initialisation d'un pointeur d'objet :

```
toto = (TypeObjet*)adresse ;
```

On initialise le pointeur avec l'adresse de notre objet de type (classe) TCanvas.



Destruction d'objets

- La commande *delete* permet de libérer l'espace mémoire occupé par les objets
- On doit l'utiliser pour détruire les objets dont on n'a plus besoin, sinon on finira par remplir toute la mémoire!

Destruction d'un objet:

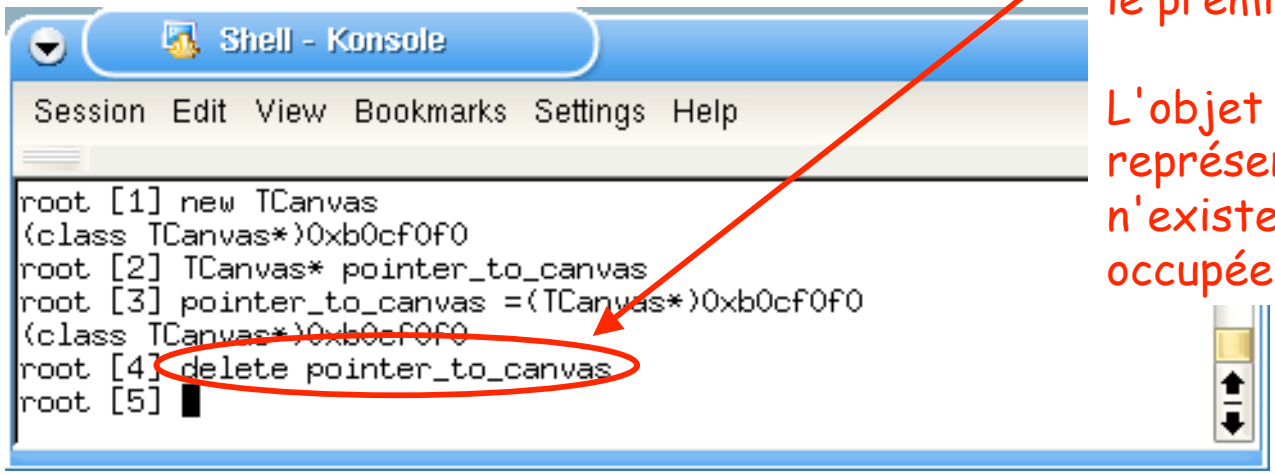
```
delete toto;
```

Destruction d'objets

- La commande *delete* permet de libérer l'espace mémoire occupé par les objets
- On doit l'utiliser pour détruire les objets dont on n'a plus besoin, sinon on finira par remplir toute la mémoire!

Destruction d'un objet:

delete toto;



L'exécution de cette commande fait disparaître le premier canevas!

L'objet (et sa représentation graphique) n'existe plus, la mémoire occupée est libérée

Les constructeurs d'objets

- La plupart du temps on effectue la déclaration du pointeur, la création de l'objet et l'initialisation du pointeur avec l'adresse de l'objet en une seule ligne!

Création d'objet avec initialisation de pointeur:

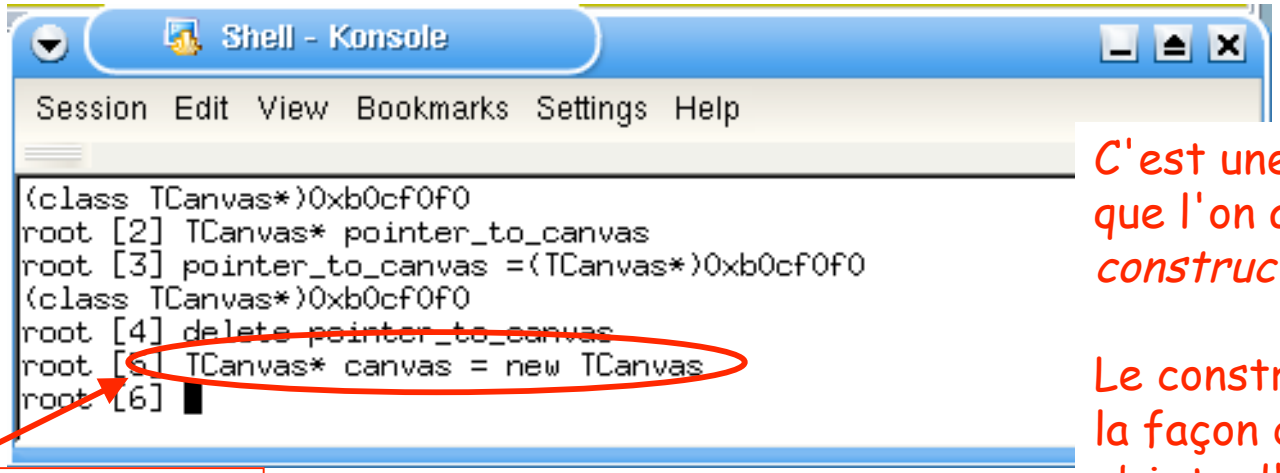
```
TypeObjet* toto = new TypeObjet;
```

Les constructeurs d'objets

- La plupart du temps on effectue la déclaration du pointeur, la création de l'objet et l'initialisation du pointeur avec l'adresse de l'objet en une seule ligne!

Création d'objet avec initialisation de pointeur:

```
TypeObjet* toto = new TypeObjet;
```



C'est une fonction spéciale que l'on appelle un *constructeur*.

Le constructeur détermine la façon de créer les objets d'une classe.

*Un autre canevas apparaît!

Les constructeurs d'objets

- En règle générale, le constructeur peut prendre des arguments

Création d'objet avec initialisation de pointeur:

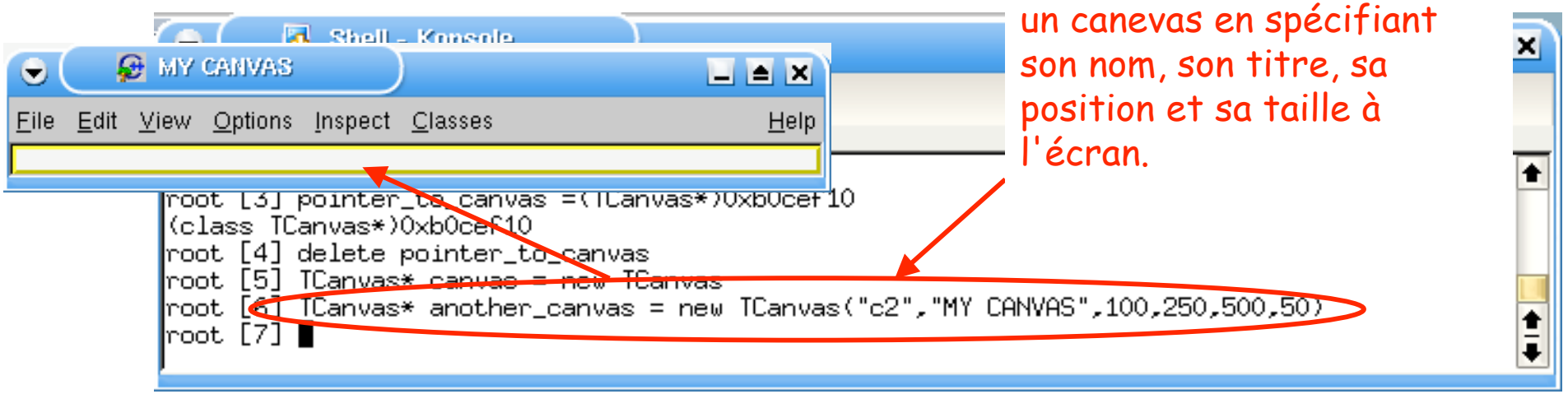
```
TypeObjet* toto = new TypeObjet(...);
```

Les constructeurs d'objets

- En règle générale, le constructeur peut prendre des arguments

Création d'objet avec initialisation de pointeur:

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



Par exemple, on peut créer un canevas en spécifiant son nom, son titre, sa position et sa taille à l'écran.

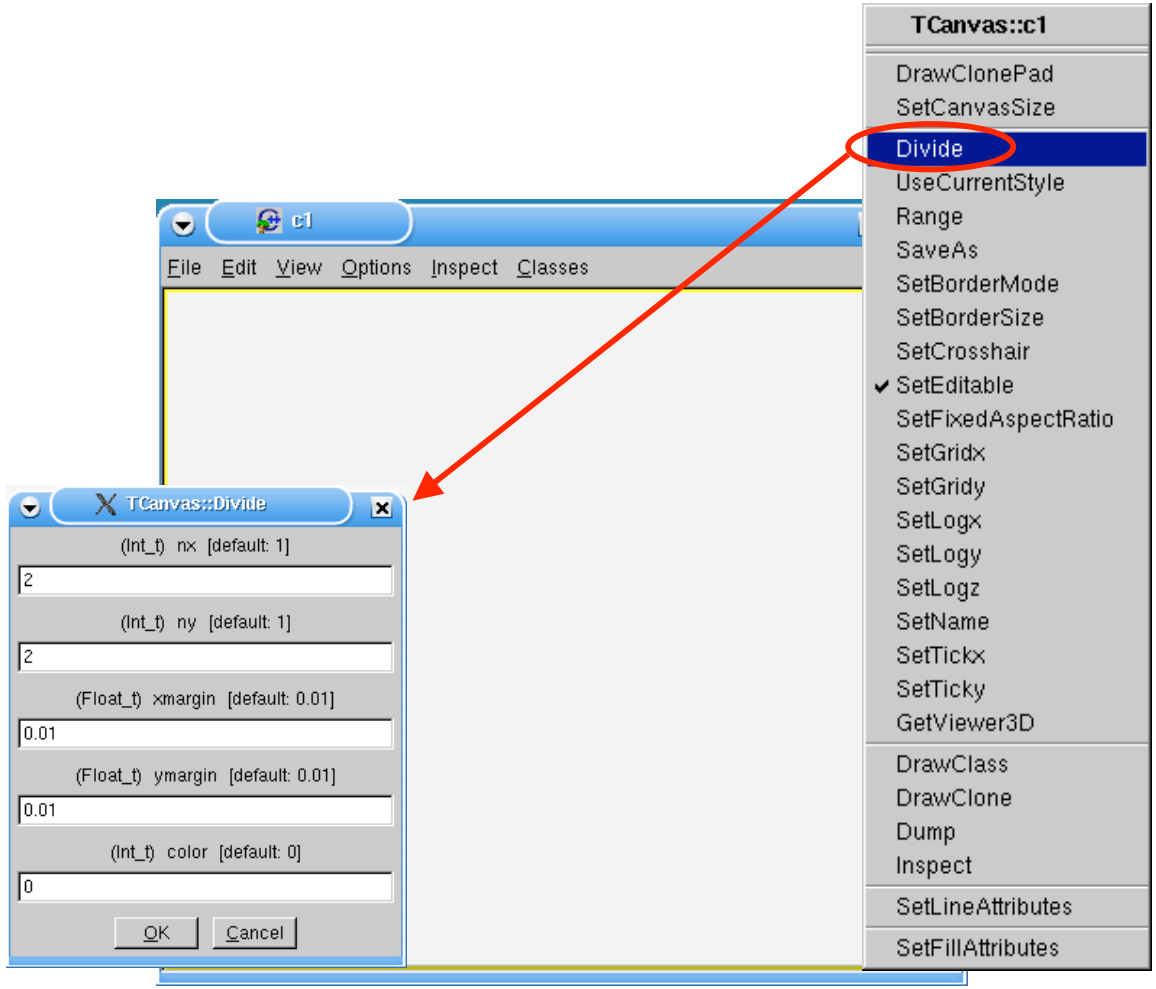
Dans l'exemple précédent, le canevas a par défaut le nom "c1" et le titre "c1"

Manipuler les objets

Les méthodes des classes

Agir sur les objets

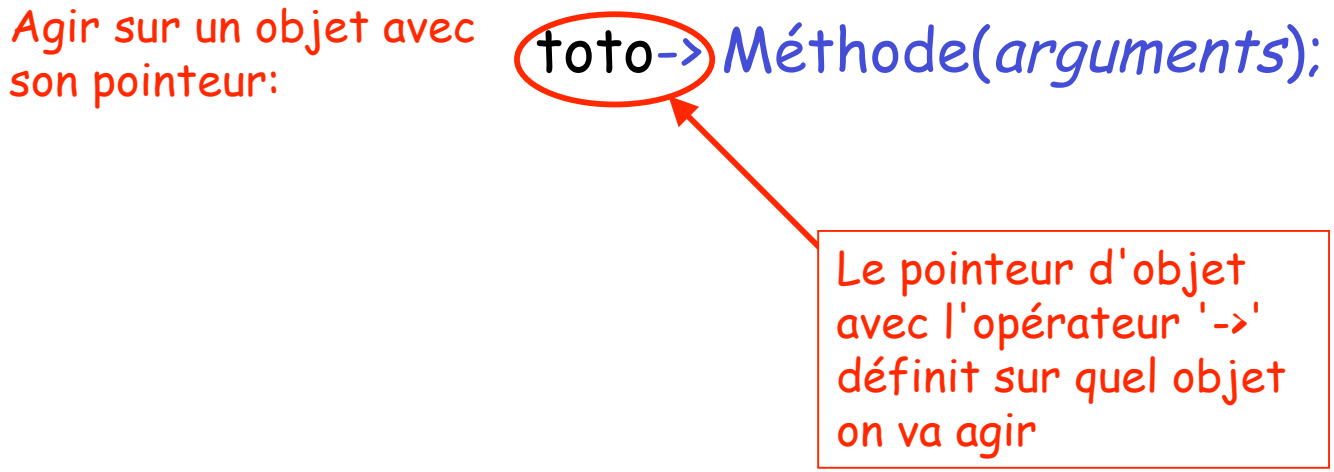
- Graphiquement, on peut agir sur un objet à travers son menu contextuel:



Exemple d'hier, quand on a divisé un canevas en 4

Les méthodes

- Avec un pointeur d'objet, on peut aussi interagir avec l'objet…



Les méthodes

- Avec un pointeur d'objet, on peut aussi interagir avec l'objet…

Agir sur un objet avec son pointeur:

```
toto-> Méthode(arguments);
```

Une des méthodes de la classe de l'objet. Par exemple, une des fonctions du menu contextuel.

Tous les objets de la même *classe* possèdent les mêmes *méthodes*.

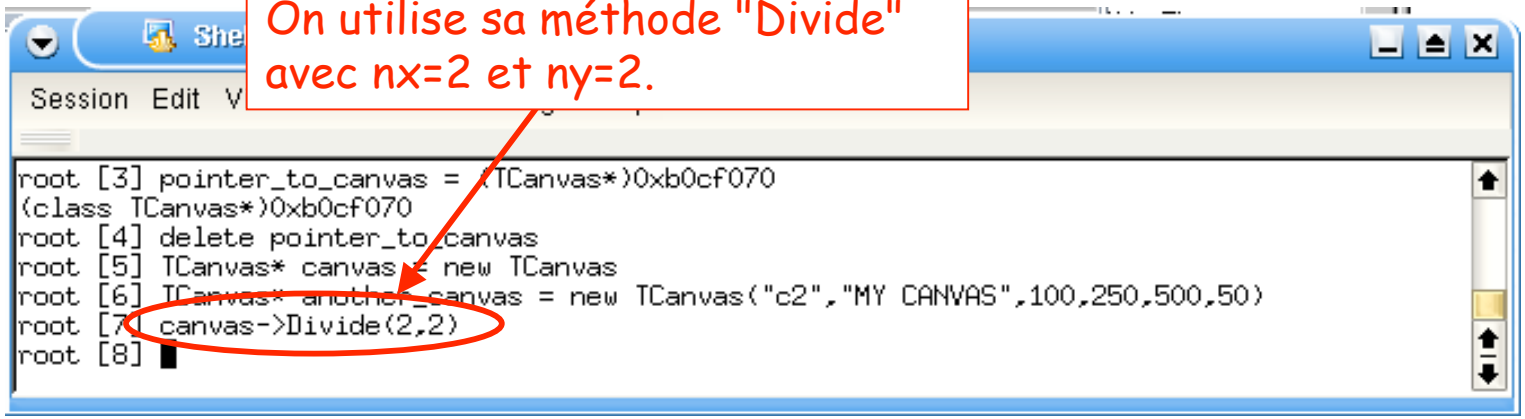
Les méthodes

- Avec un pointeur d'objet, on peut aussi interagir avec l'objet…

Agir sur un objet avec son pointeur:

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

On agit sur l'objet "c1" à travers son pointeur "canvas". On utilise sa méthode "Divide" avec nx=2 et ny=2.

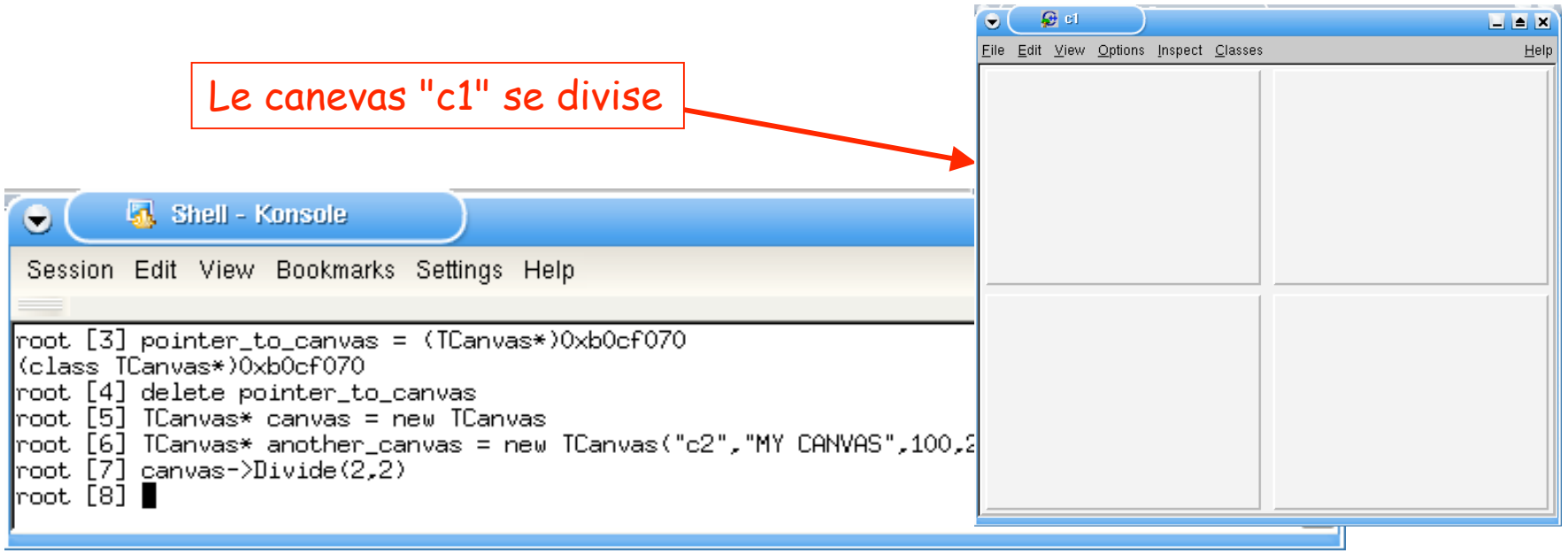


Les méthodes

- Avec un pointeur d'objet, on peut aussi interagir avec l'objet…

Agir sur un objet avec son pointeur:

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



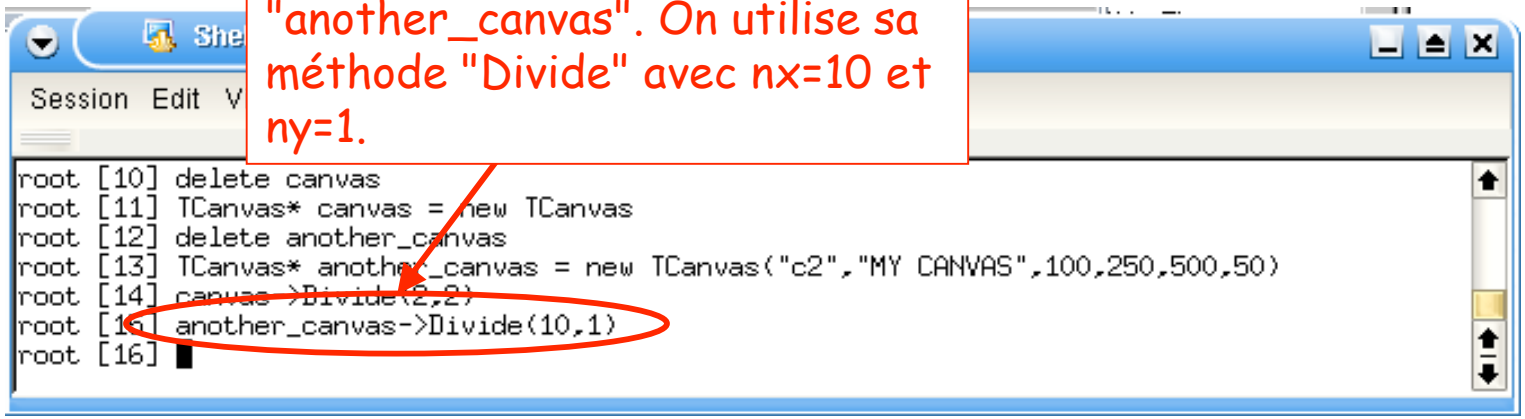
Les méthodes

- Avec un pointeur d'objet, on peut aussi interagir avec l'objet…

Agir sur un objet avec son pointeur:

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

On agit sur l'objet "c2" à travers son pointeur "another_canvas". On utilise sa méthode "Divide" avec nx=10 et ny=1.



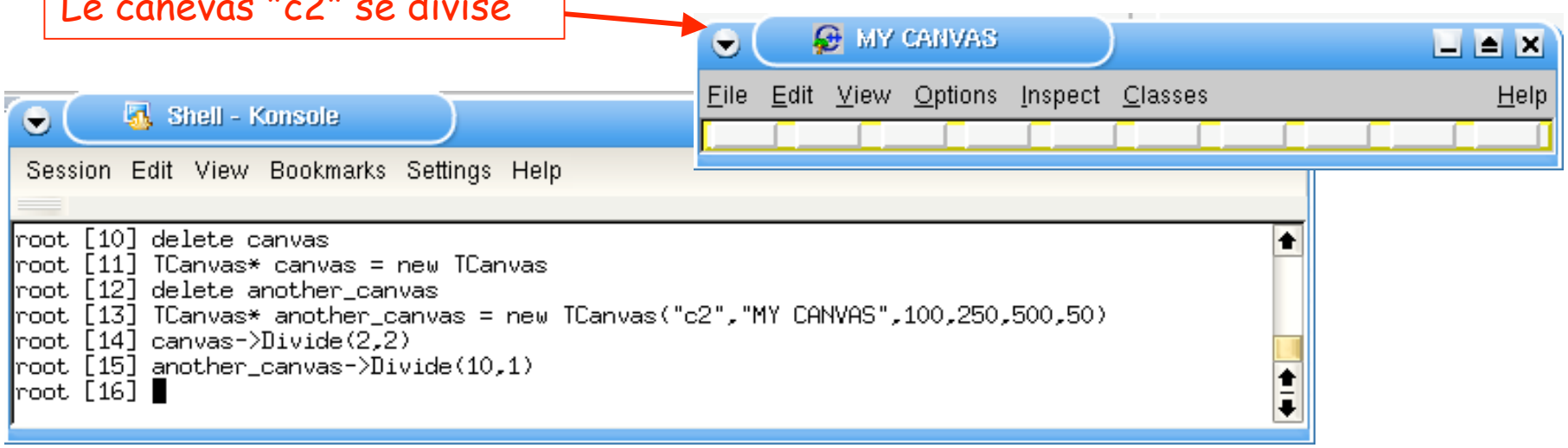
Les méthodes

- Avec un pointeur d'objet, on peut aussi interagir avec l'objet…

Agir sur un objet avec son pointeur:

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

Le canevas "c2" se divise

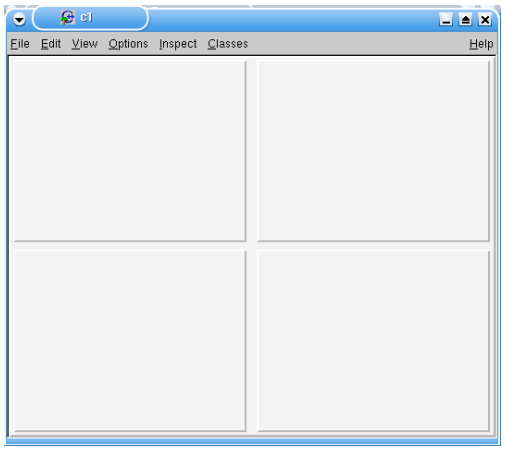


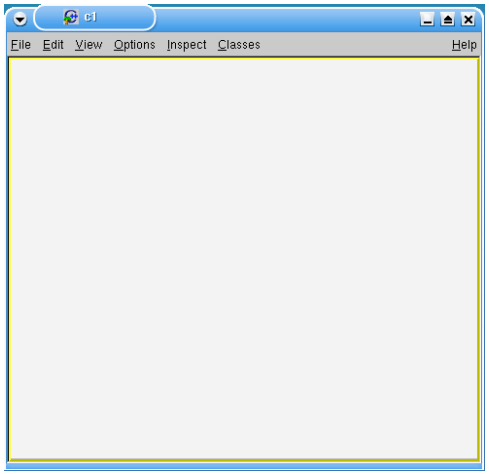
Les méthodes

- D'autres opérations sur les canevas:

```
canvas->Clear();
```

Effacer le contenu du canevas, y compris les divisions





Rendre le canevas "actif", c-à-d c'est lui qui aura un pourtour jaune, et le prochain histo s'affichera la-dessus

```
canvas->cd();
```

Exemple avec un histogramme

- On peut créer un spectre à une dimension de la même façon que pour les canevas:

Création d'un histo 1D

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

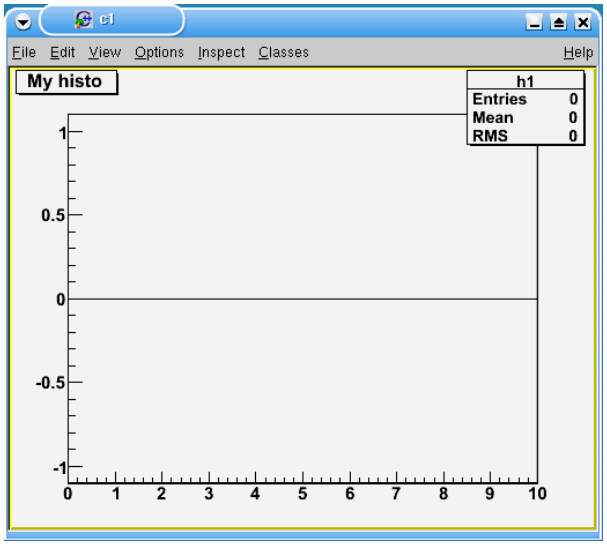
```
TypeObjet* toto = new TypeObjet(...);
```

REMARQUE: L'histo ne s'affiche pas automatiquement!

TH1F ? Histo à 1D de nombres réels (Float)

Exemple avec un histogramme

- Afficher et remplir l'histogramme:



Afficher un spectre: `histo->Draw();`

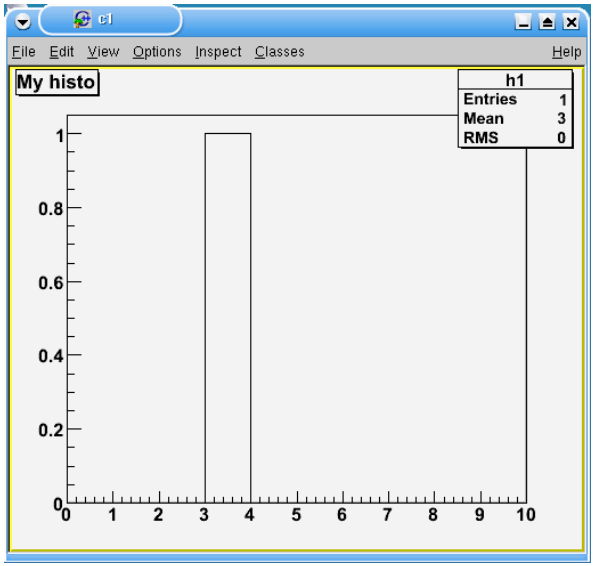
Remplir un spectre: `histo->Fill(3);`

L'argument "3" correspond à une valeur de l'abscisse

Il ne s'est rien passé ?

Exemple avec un histogramme

- Mettre à jour l'affichage:



Signaler au canevas qu'un objet affiché a été modifié:

```
canvas->Modified();
```

A la ligne de commande, ça provoque automatiquement la mise à jour du canevas.

On verra par la suite que la plupart du temps (e.g. dans un programme) il faut aussi demander la mise à jour!

Forcer la mise à jour du canevas:

```
canvas->Update();
```

Exemple avec un histogramme

- On commence à s'ennuyer?

On pourrait continuer comme ça jusqu'à ce que notre spectre soit rempli…

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

Remplissage avec un poids

Mais on ferait mieux d'écrire une boucle…
…dans une fonction…
…en C++
…!!!

Création d'objets sans "new"

Il y a une autre façon de faire

Objets "temporaires" vs. objets "permanents"

L'autre façon de faire…

- Il y a une autre façon de créer et de manipuler les objets…

Création d'objet sans "new"

```
TypeObjet toto(…);
```

L'autre façon de faire…

- Il y a une autre façon de créer et de manipuler les objets…

Création d'objet sans "new"

```
TypeObjet toto(…);
```

Si le constructeur prend des arguments, on les met ici

Création d'objet avec "new"

```
TypeObjet* toto_ptr = new TypeObjet(…);
```

L'autre façon de faire…

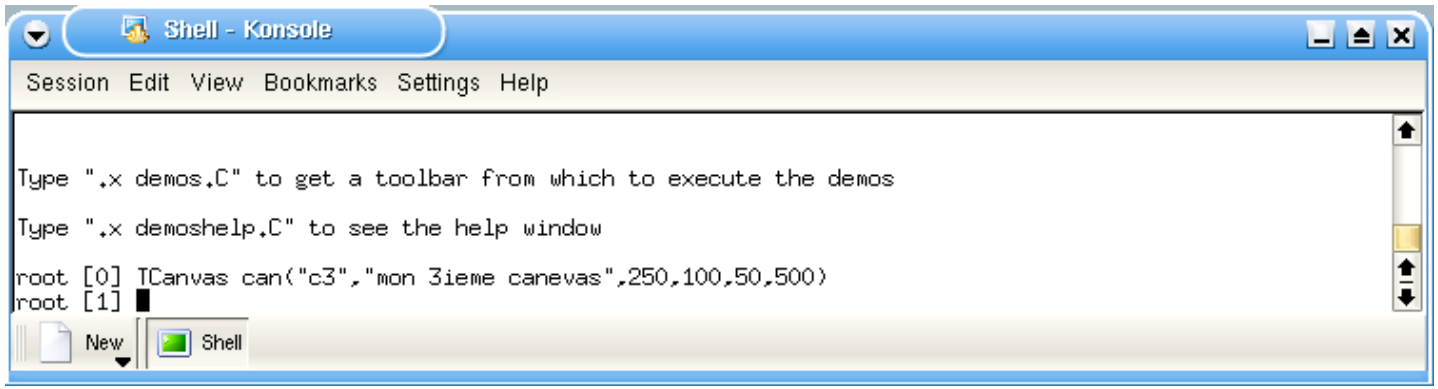
- Il y a une autre façon de créer et de manipuler les objets…

Création d'objet sans "new"

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

Création d'objet avec "new"

```
TCanvas* can_ptr = new TCanvas("c3","titre", …);
```

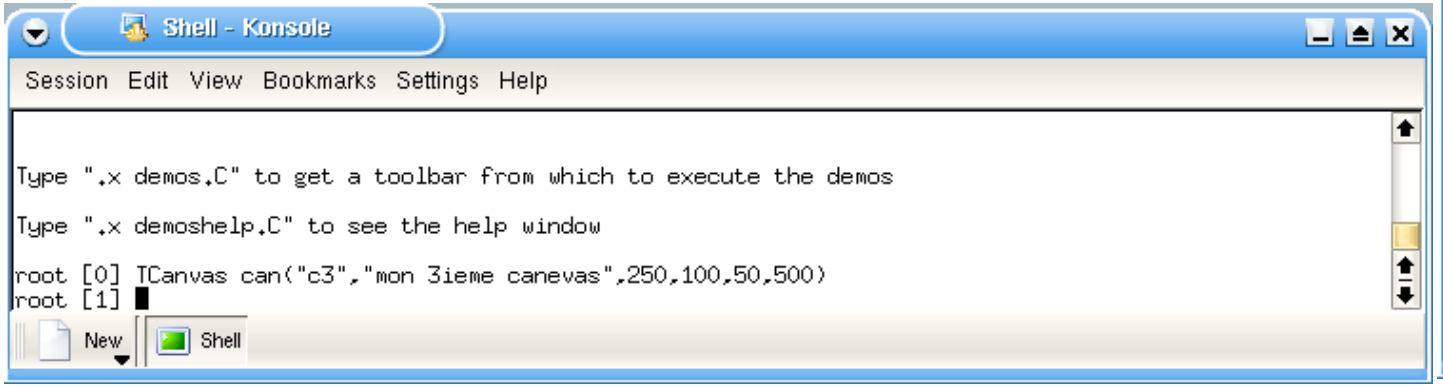


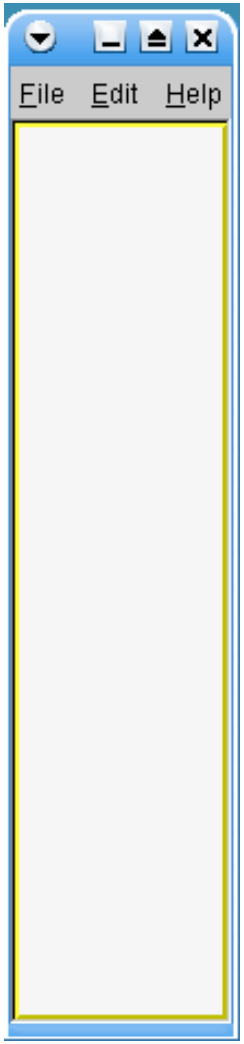
L'autre façon de faire…

- La façon d'agir sur l'objet n'est plus tout à fait la même…

Création d'objet sans "new"

`toto.` *Méthode(arguments);*





L'autre façon de faire…

- La façon d'agir sur l'objet n'est plus tout à fait la même…

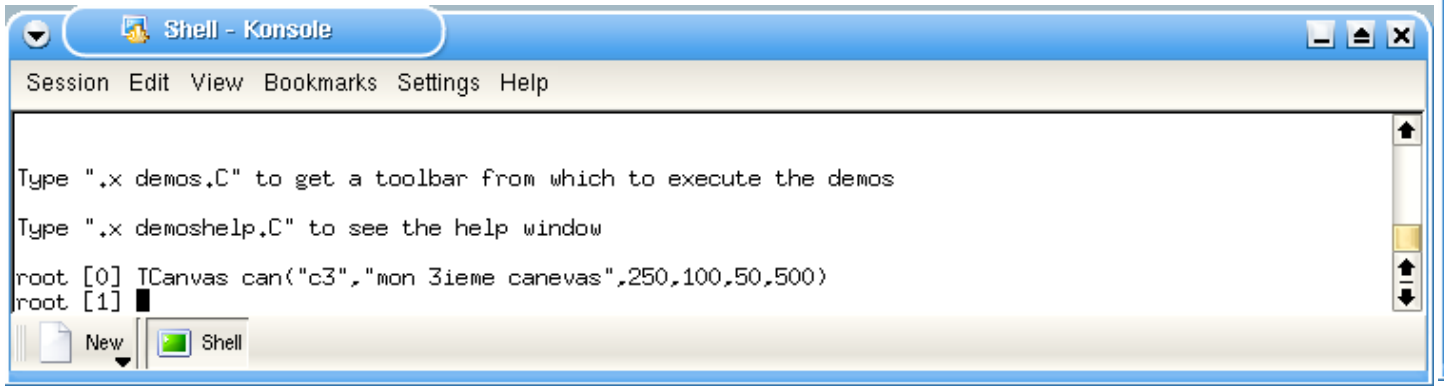
Création d'objet sans "new"

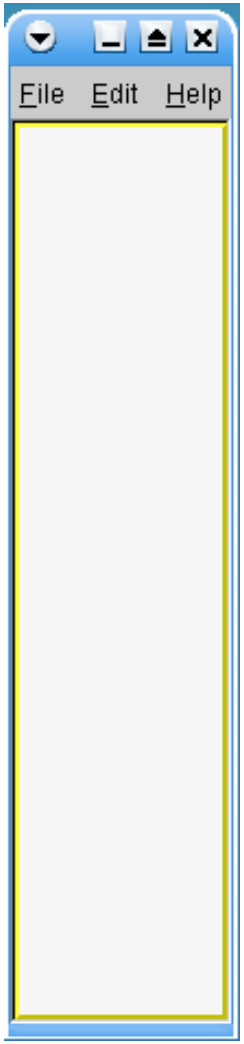
toto.Méthode(*arguments*);

Dans un cas on utilise un point, dans l'autre une flèche

Agir sur un objet avec son pointeur:

toto_ptr->Méthode(*arguments*);





L'autre façon de faire…

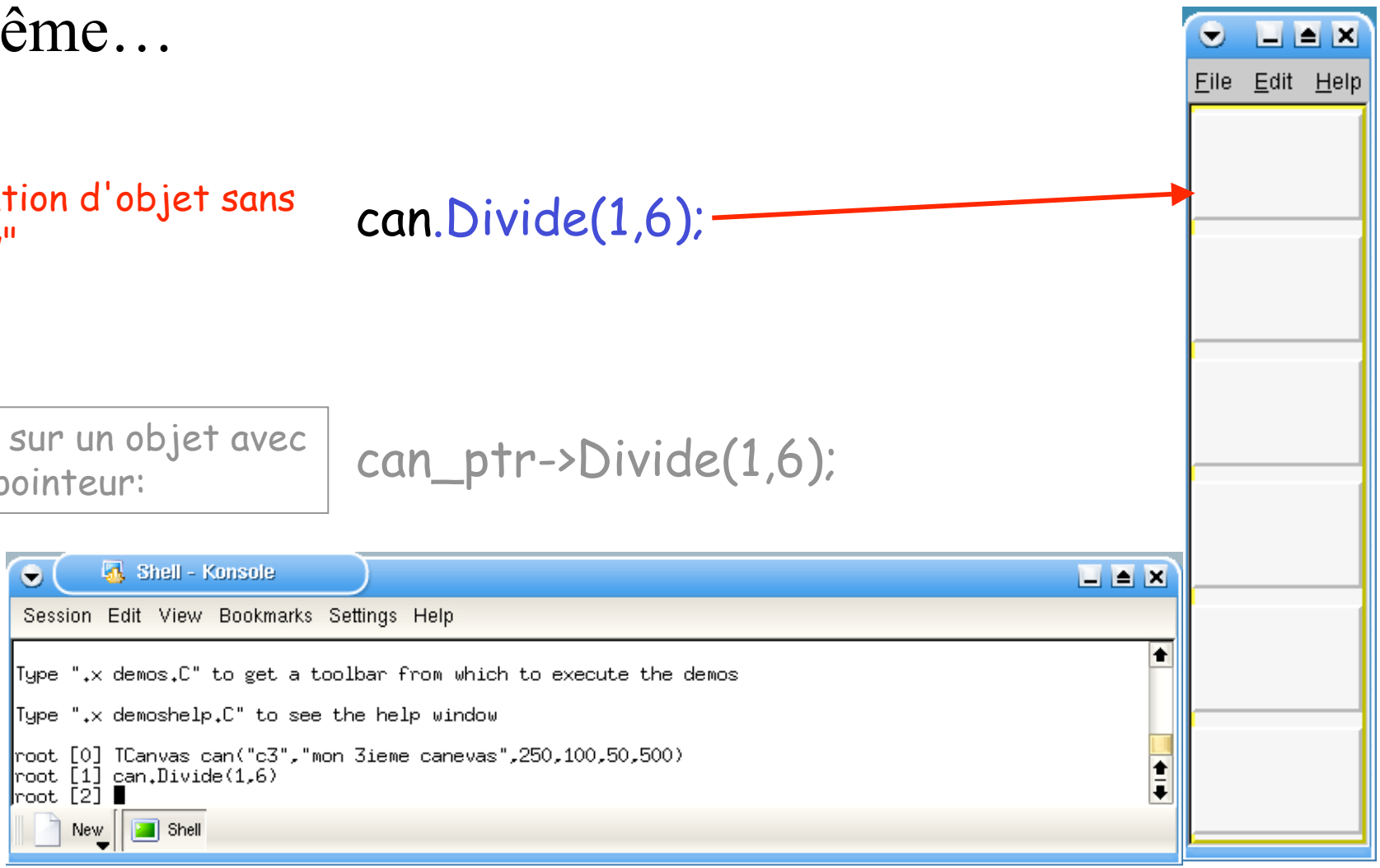
- La façon d'agir sur l'objet n'est plus tout à fait la même…

Création d'objet sans "new"

```
can.Divide(1,6);
```

Agir sur un objet avec son pointeur:

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



L'autre façon de faire…

- On peut aussi obtenir l'adresse mémoire des objets créés de cette façon, et les manipuler en utilisant un *pointeur*

Initialiser un pointer avec l'adresse d'un objet existant

```
TypeObjet* toto_ptr = &toto;
```

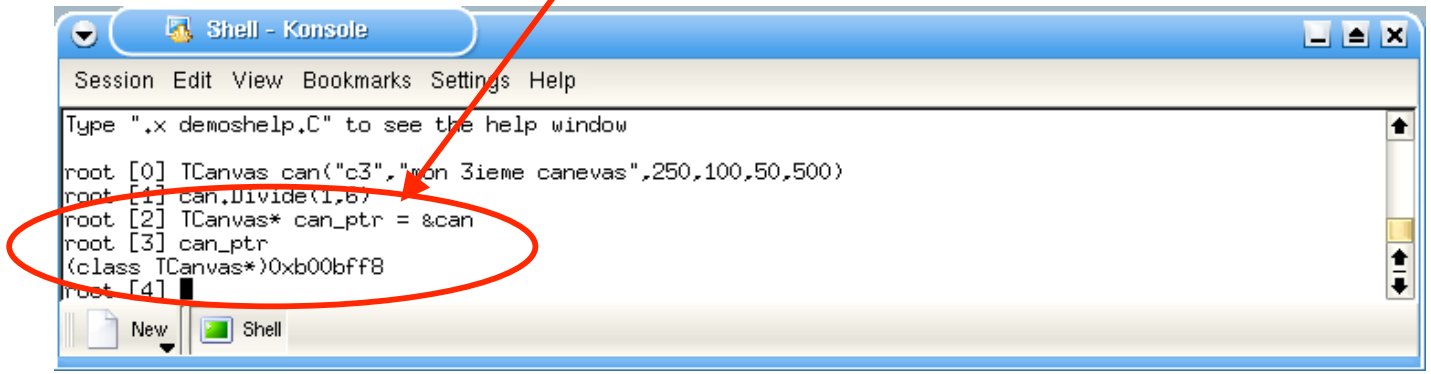
L'autre façon de faire…

- On peut aussi obtenir l'adresse mémoire des objets créés de cette façon, et les manipuler en utilisant un *pointeur*

Initialiser un pointer avec l'adresse d'un objet existant

```
TypeObjet* toto_ptr = &toto;
```

Opérateur qui renvoie l'adresse mémoire de l'objet

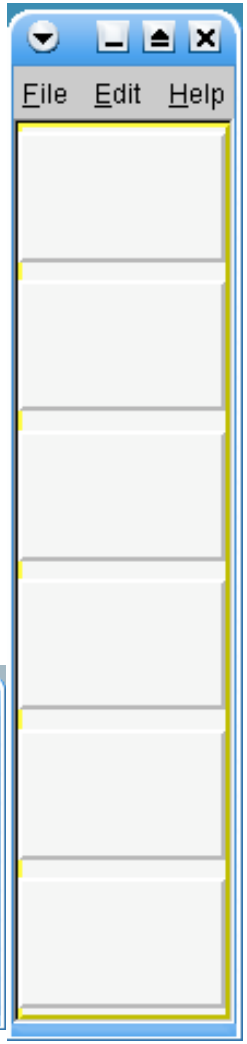


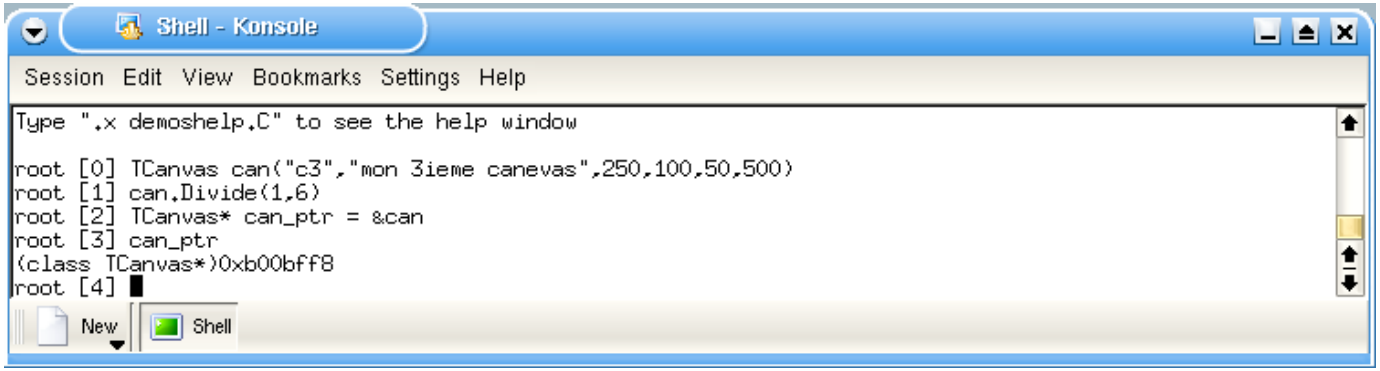
L'autre façon de faire…

- L'utilisation du *pointeur* pour agir sur l'objet ensuite est identique aux cas précédents…

Agir sur l'objet à travers son pointeur

```
toto_ptr ->Méthode(arguments);
```



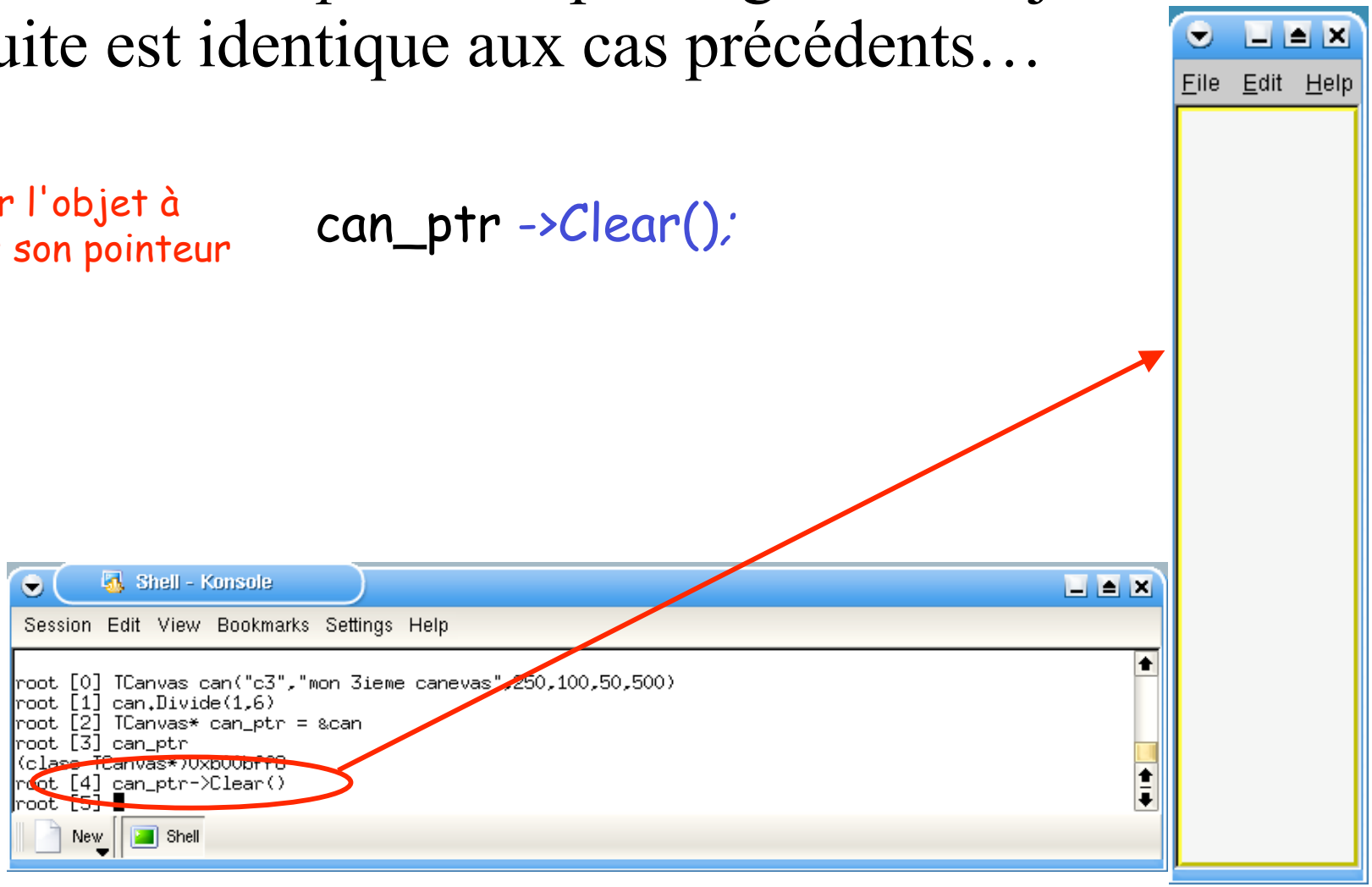


L'autre façon de faire…

- L'utilisation du *pointeur* pour agir sur l'objet ensuite est identique aux cas précédents…

Agir sur l'objet à travers son pointeur

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

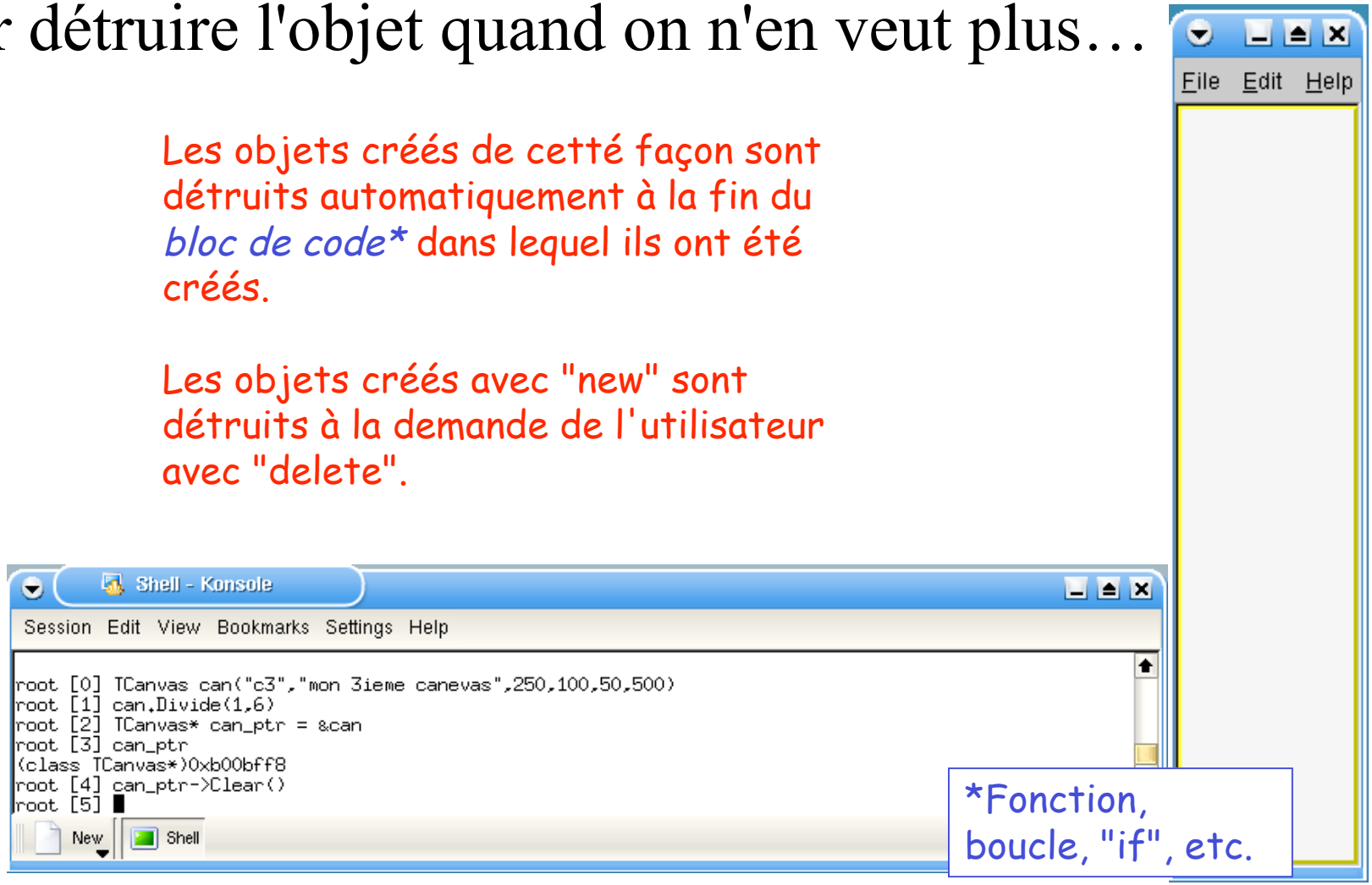


L'autre façon de faire…

- La différence ? On n'a pas besoin d'utiliser *delete* pour détruire l'objet quand on n'en veut plus…

Les objets créés de cetté façon sont détruits automatiquement à la fin du *bloc de code** dans lequel ils ont été créés.

Les objets créés avec "new" sont détruits à la demande de l'utilisateur avec "delete".



L'autre façon de faire…

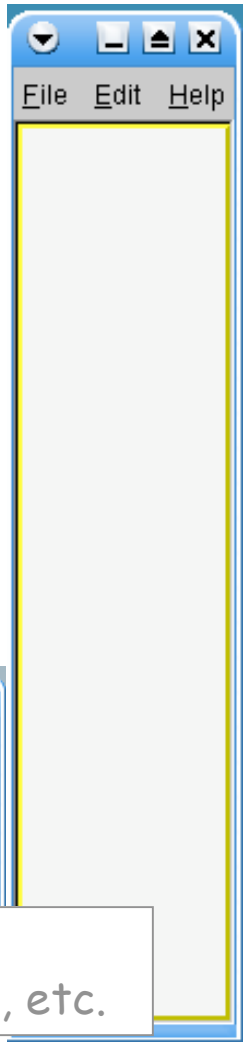
- La différence ? On n'a pas besoin d'utiliser *delete* pour détruire l'objet quand on n'en veut plus…

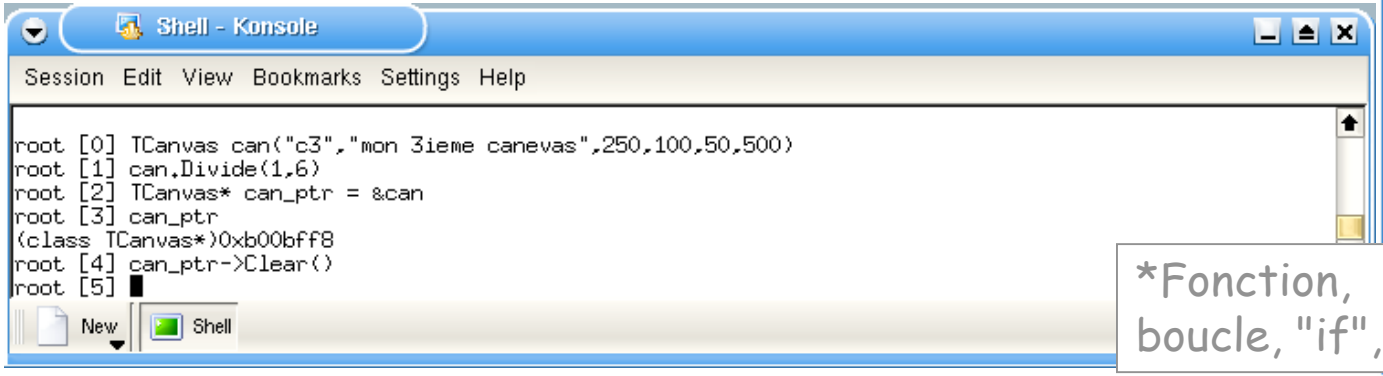
Objets temporaires

Les objets créés de cette façon sont détruits automatiquement à la fin du *bloc de code** dans lequel ils ont été créés.

Objets permanents

Les objets créés avec "new" sont détruits à la demande de l'utilisateur avec "delete".





*Fonction, boucle, "if", etc.

Trouver l'information sur les classes

Où est le mode d'emploi ?

Faut-il tout apprendre par cœur ?*

*NON!!!!

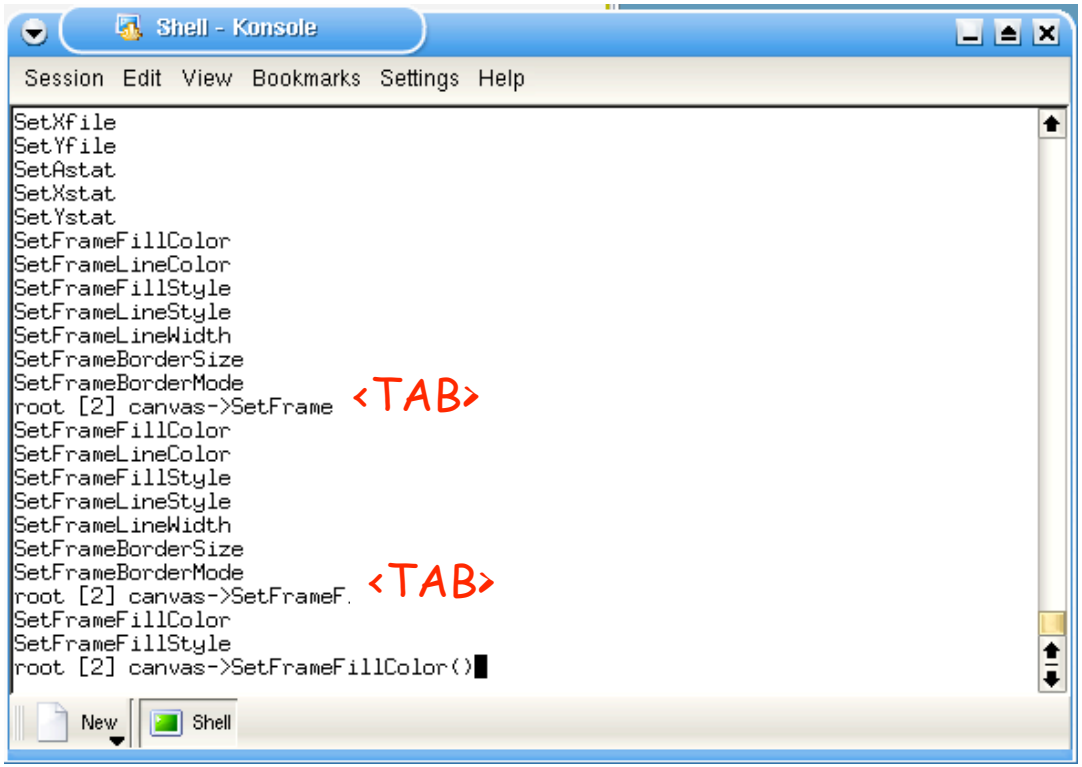
Trouver l'information

- Mais comment connaître toutes les façons d'interagir avec un objet ?
- Comment connaître toutes les méthodes d'une classe ?

1. la complétion des commandes avec la touche <TAB>

Très efficace, permet de réduire au minimum les commandes à taper soi-même (c-à-d le nombre possible de fautes de frappe)...

ASTUCE:
la plupart des méthodes qui modifie un objet commencent par "Set..."
la plupart des méthodes qui renseignent sur un objet commencent par "Get..."

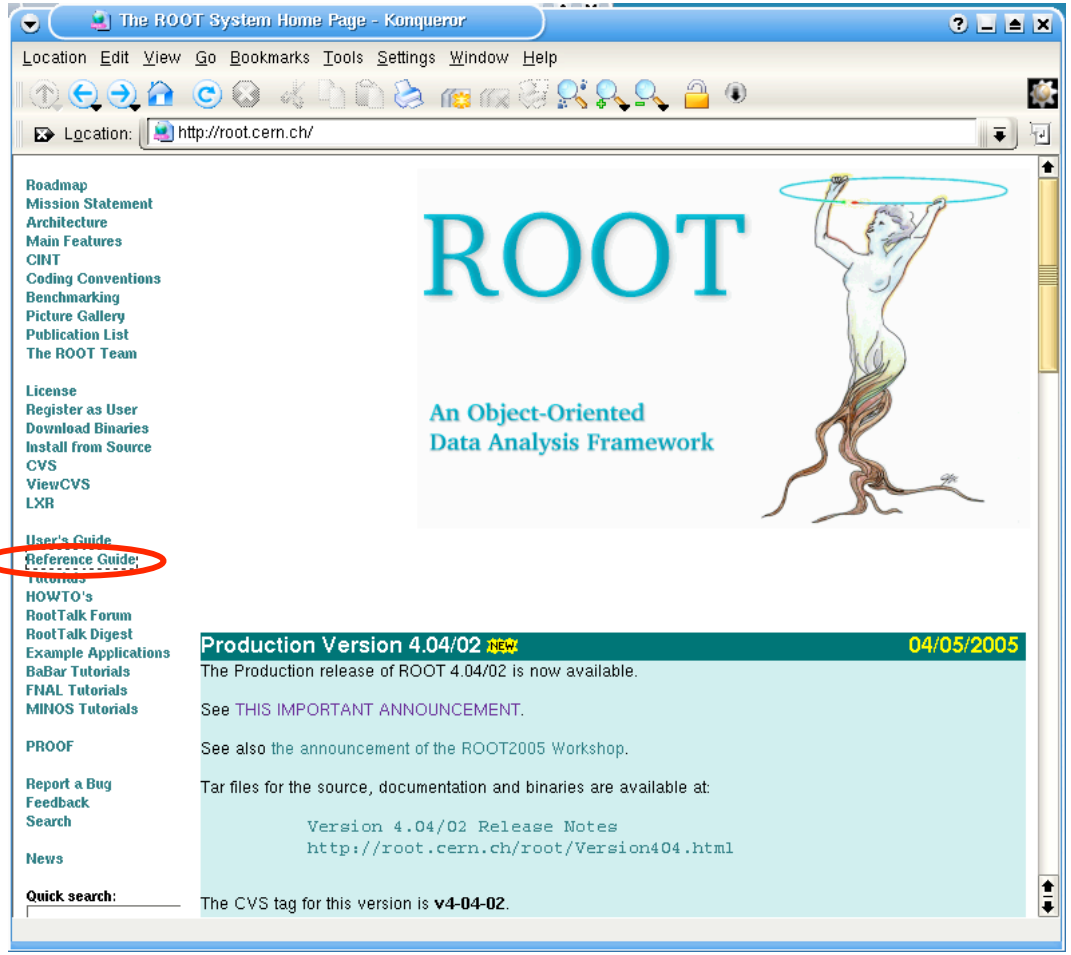


Trouver l'information

- La meilleur façon de se renseigner: consulter le site web http://root.cern.ch

2. Regarder le site web

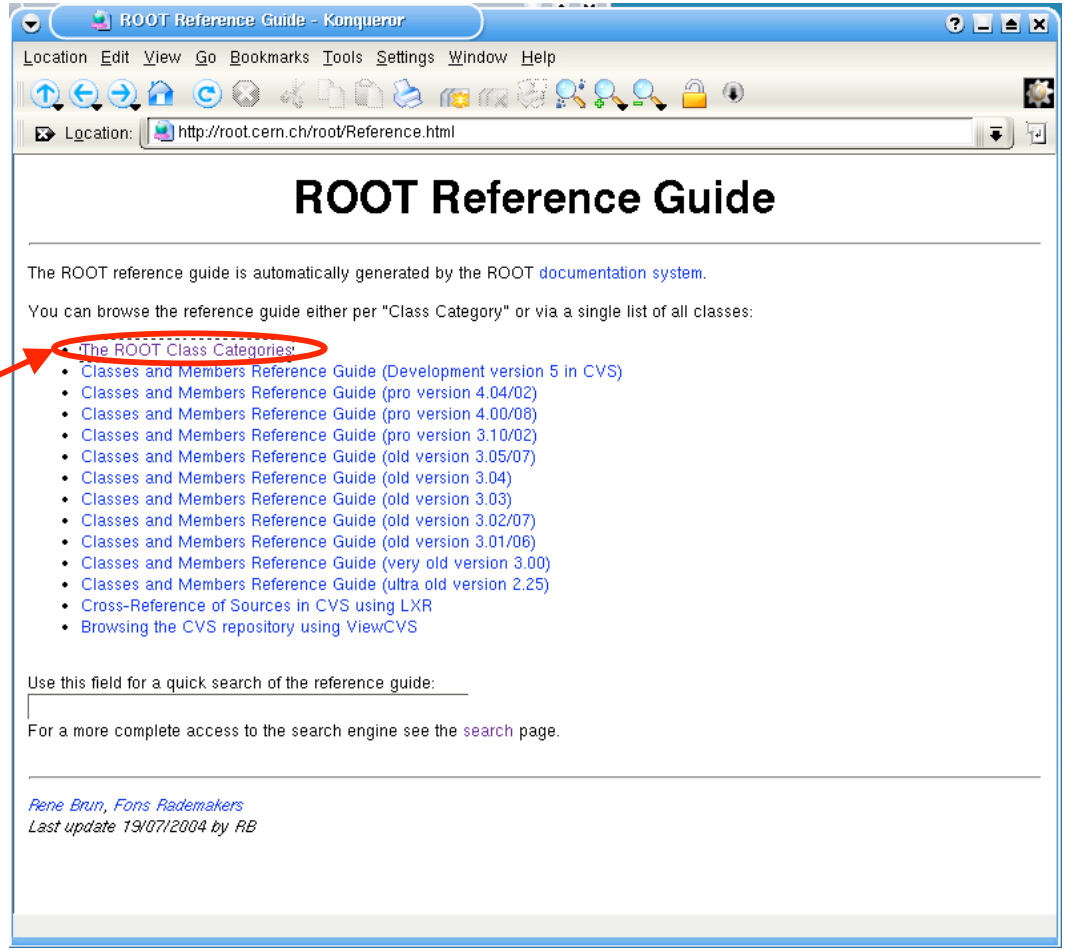
1. clique sur "Reference Guide"



Consultation du site web

- La meilleur façon de se renseigner: consulter le site web http://root.cern.ch

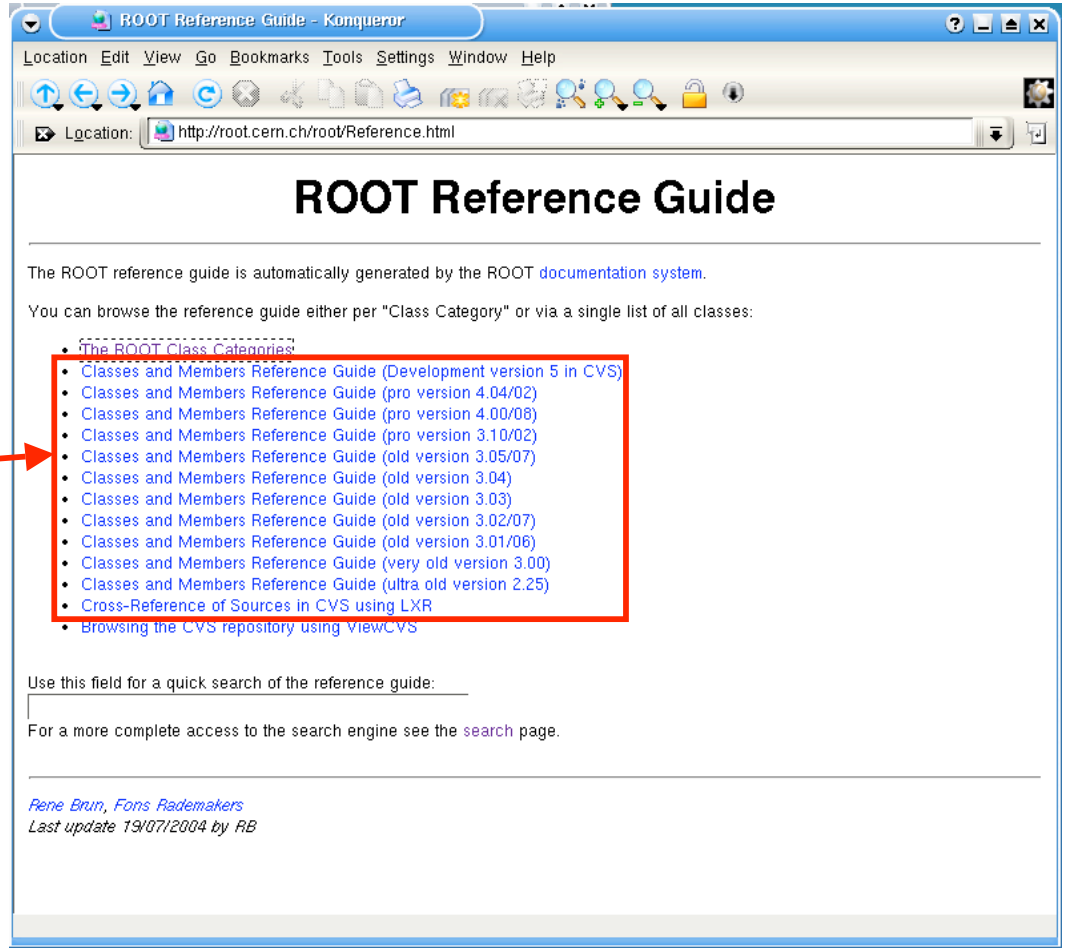
2. liste des classes par catégories (histos, matrices, géométrie, etc.)



Consultation du site web

- La meilleur façon de se renseigner: consulter le site web http://root.cern.ch

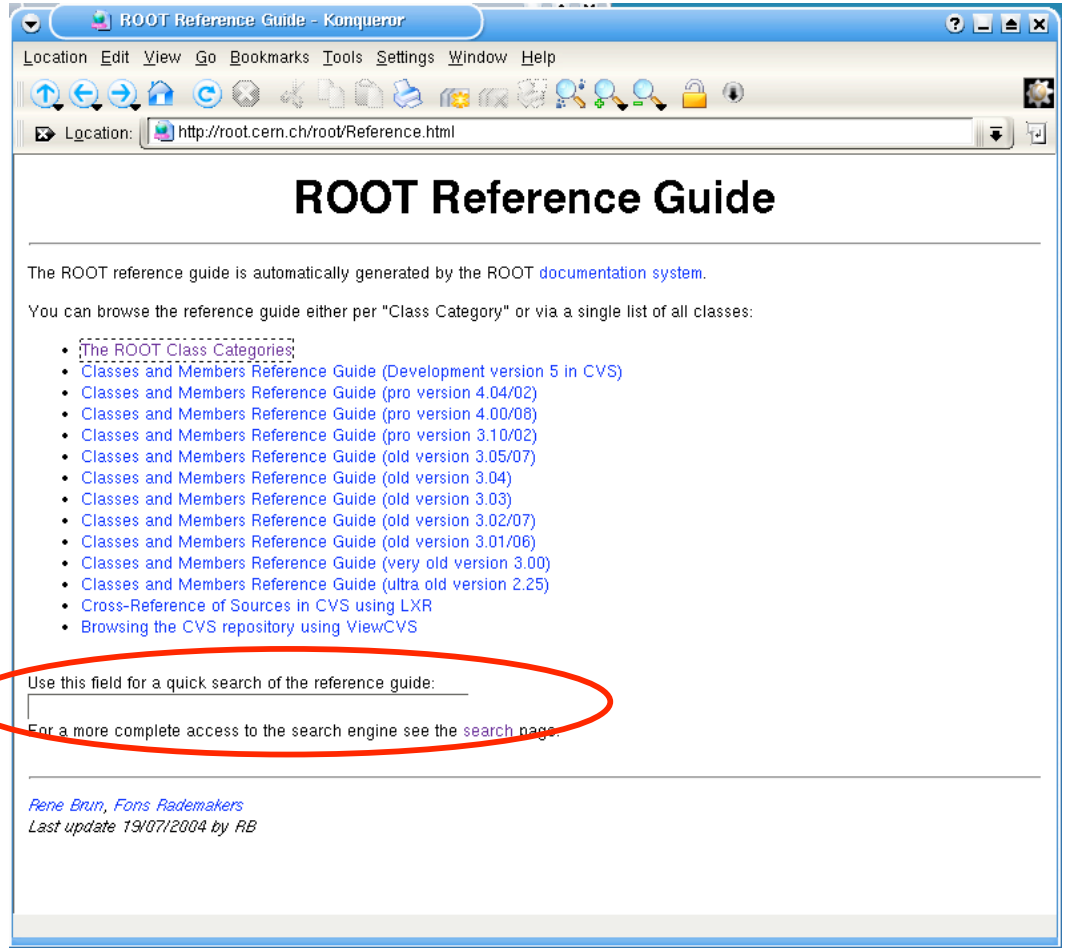
2. liste complète des classes pour chaque version de ROOT



Consultation du site web

- La meilleur façon de se renseigner: consulter le site web http://root.cern.ch

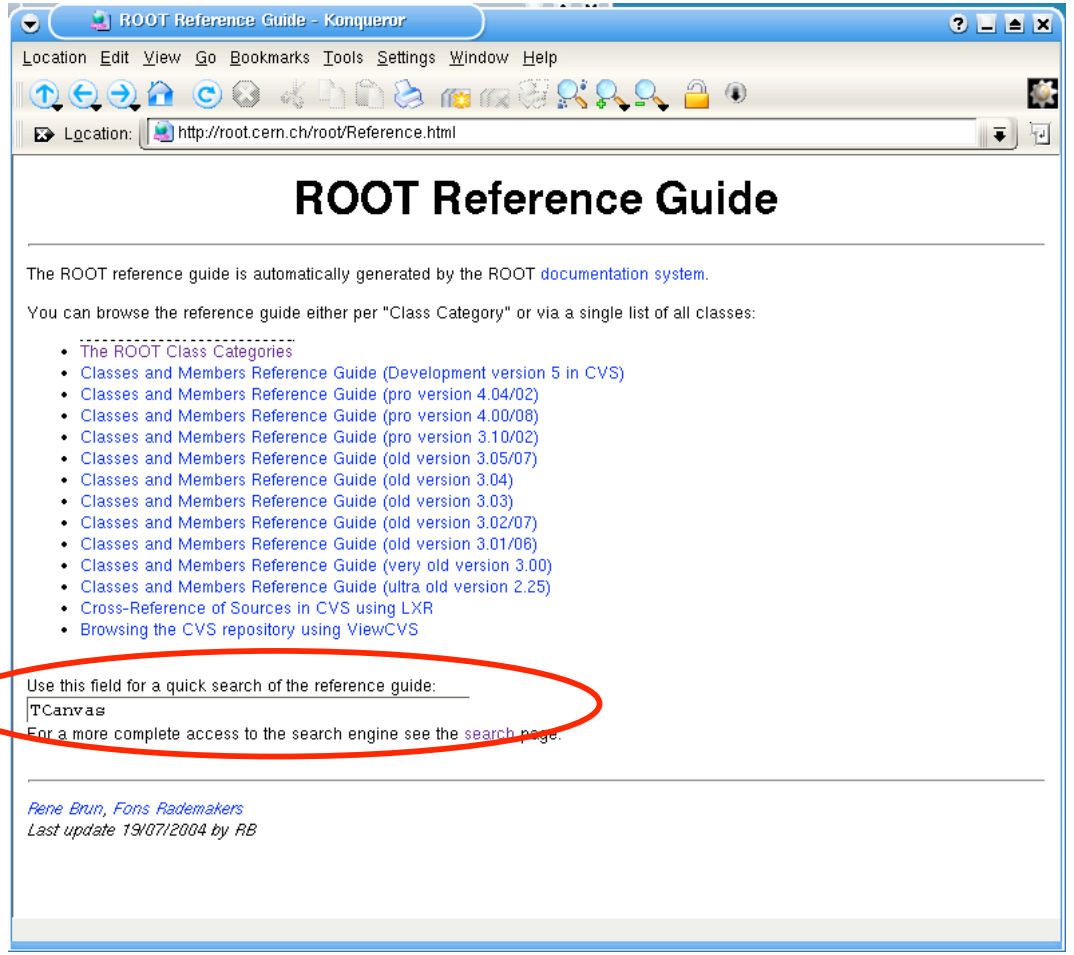
2. recherche sur mots clés



Consultation du site web

- La meilleur façon de se renseigner: consulter le site web http://root.cern.ch

rentrez
"TCanvas"
(puis <ENTER>)

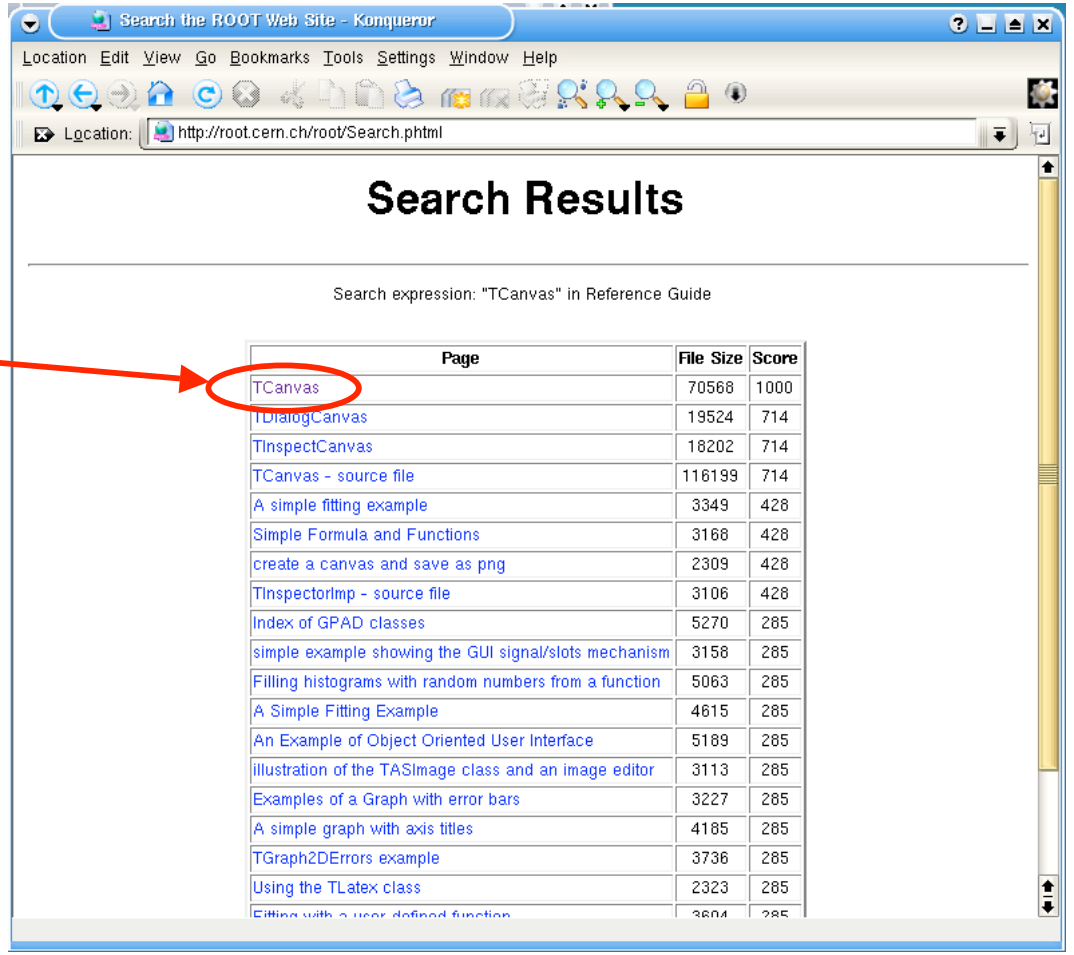


Consultation du site web

- La meilleur façon de se renseigner: consulter le site web http://root.cern.ch

la recherche par nom de classe marche le mieux…

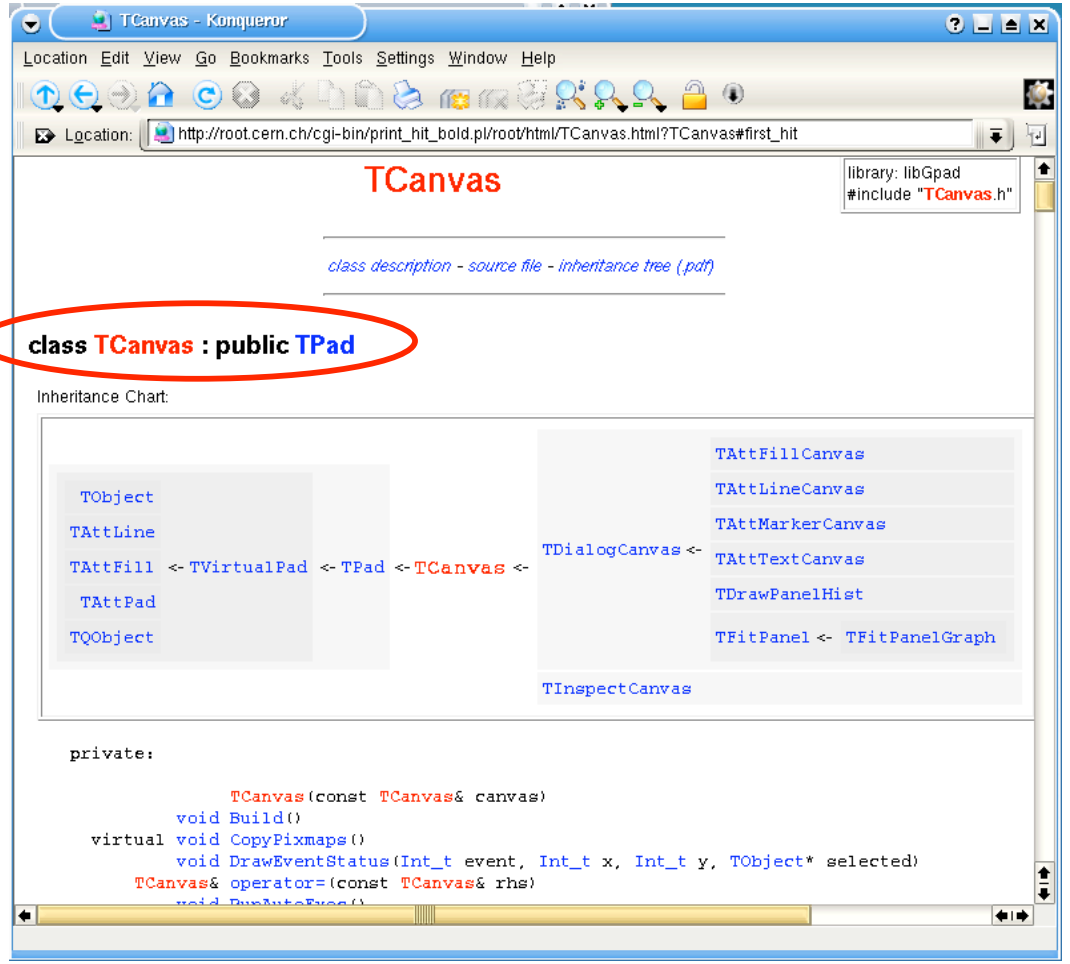
…mais ne négligez pas les autres résultats qui peuvent être très intéressants!!



Présentation d'une page de description d'une classe

- On y trouve toutes les informations nécessaires… à condition de savoir les trouver

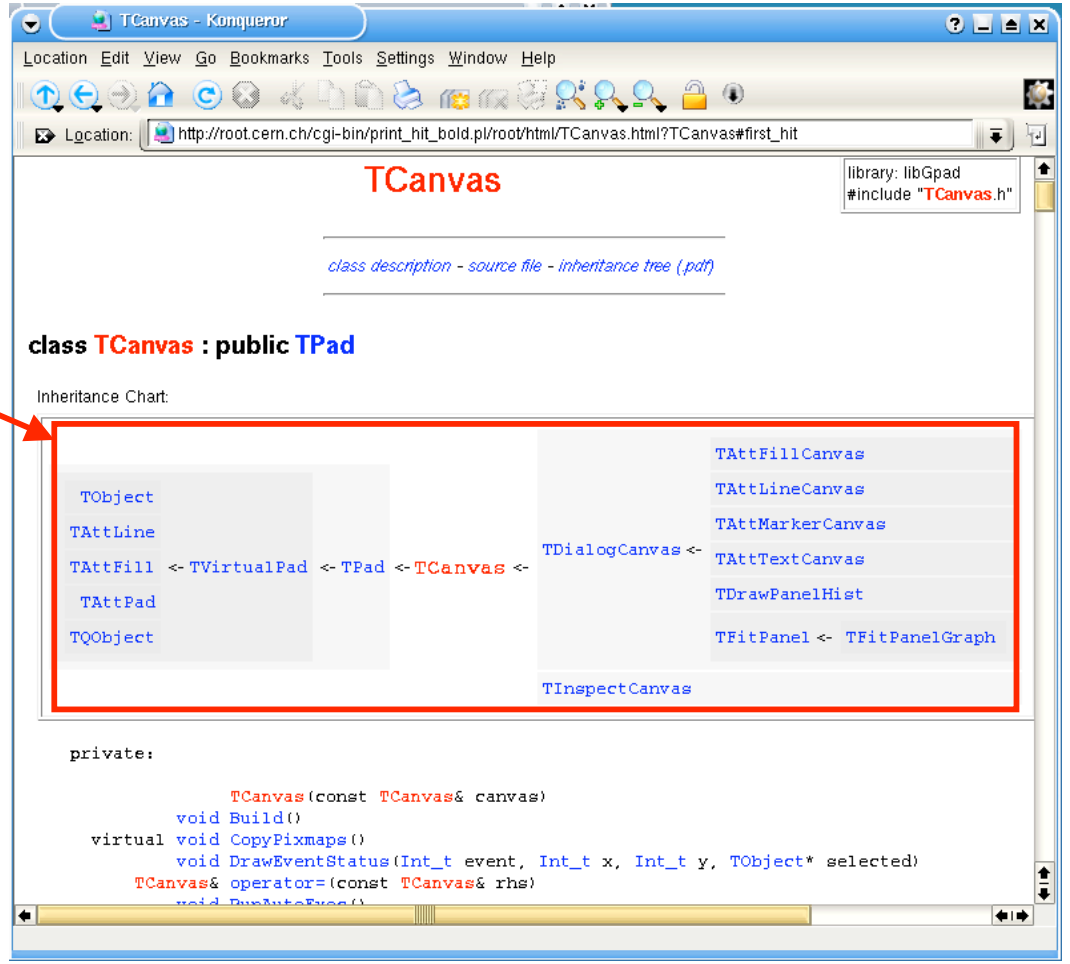
Le nom de la classe et de son parent le plus proche (classe "mère")



Présentation d'une page de description d'une classe

- On y trouve toutes les informations nécessaires… à condition de savoir les trouver

Généalogie complète de la classe

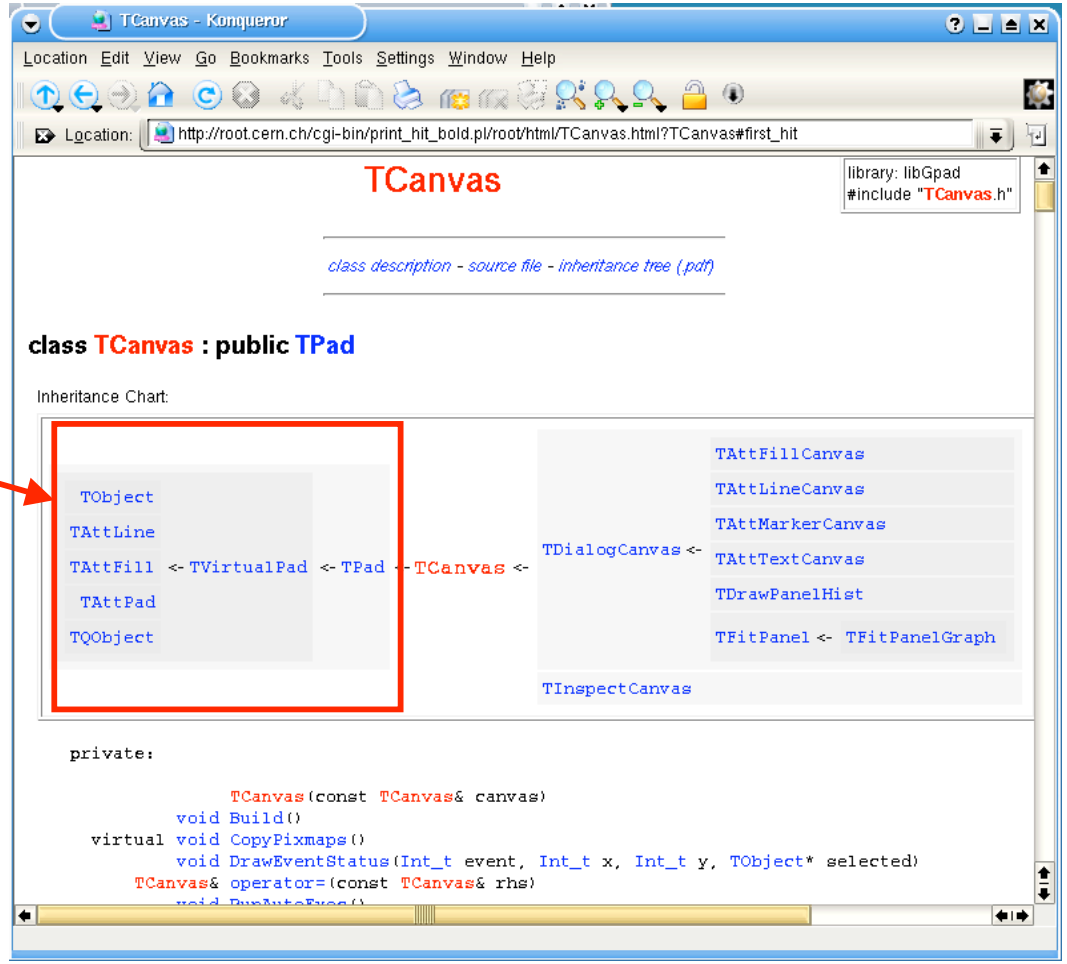


Présentation d'une page de description d'une classe

- On y trouve toutes les informations nécessaires… à condition de savoir les trouver

Généalogie complète de la classe

parents et grand-parents à gauche



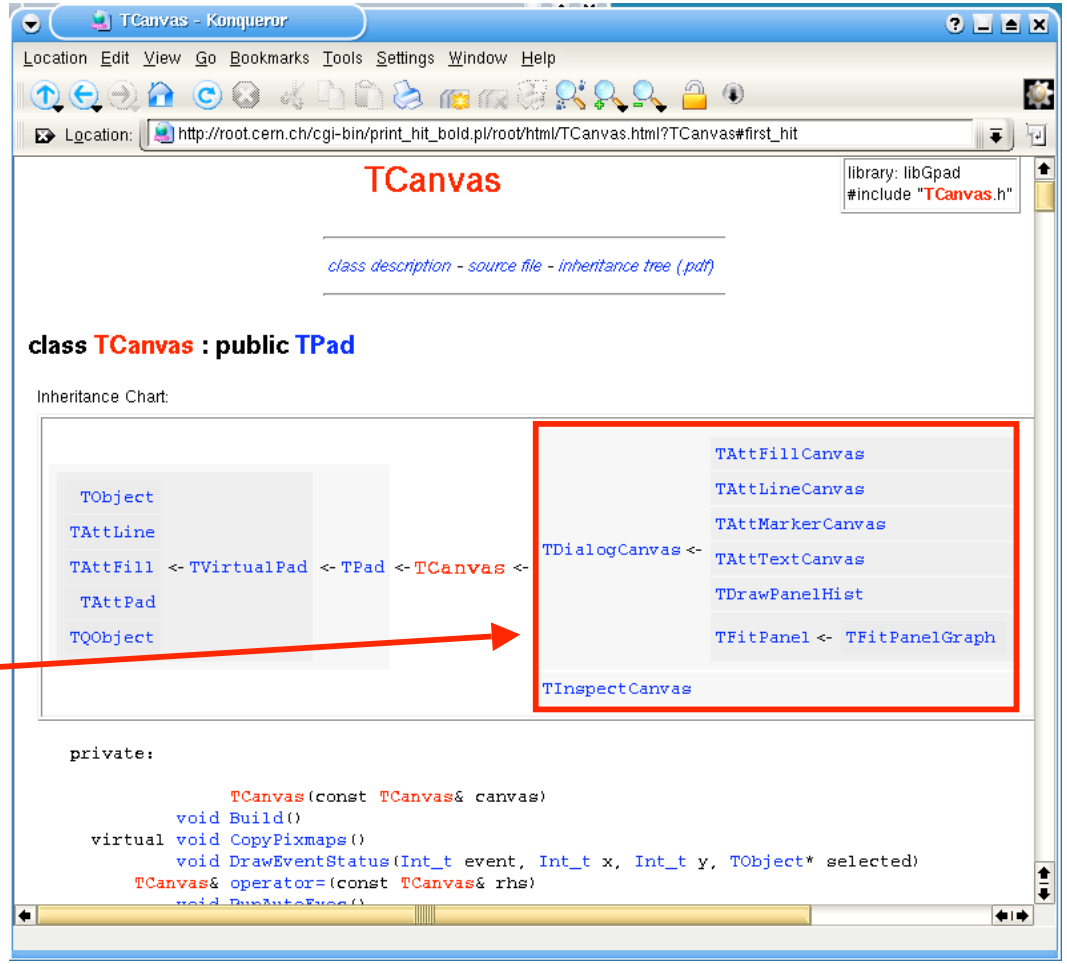
Présentation d'une page de description d'une classe

- On y trouve toutes les informations nécessaires… à condition de savoir les trouver

Généalogie complète de la classe

parents et grand-parents à gauche

enfants et petit-enfants à droite



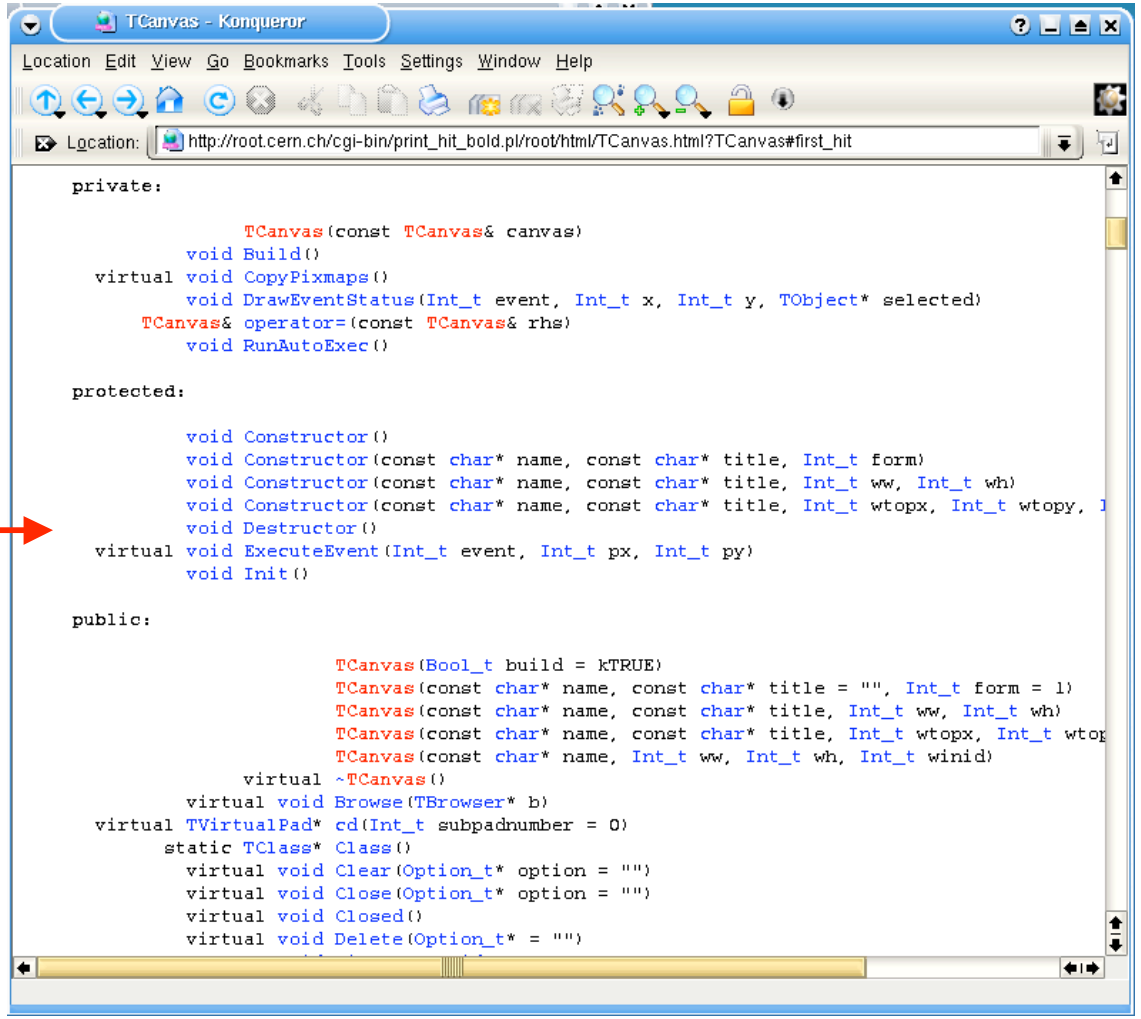
La liste des méthodes d'une classe

- On y trouve toutes les informations nécessaires… à condition de savoir les trouver

(un peu plus bas)

la liste complète des méthodes de la classe

chaque nom est un lien vers une documentation complète de la méthode

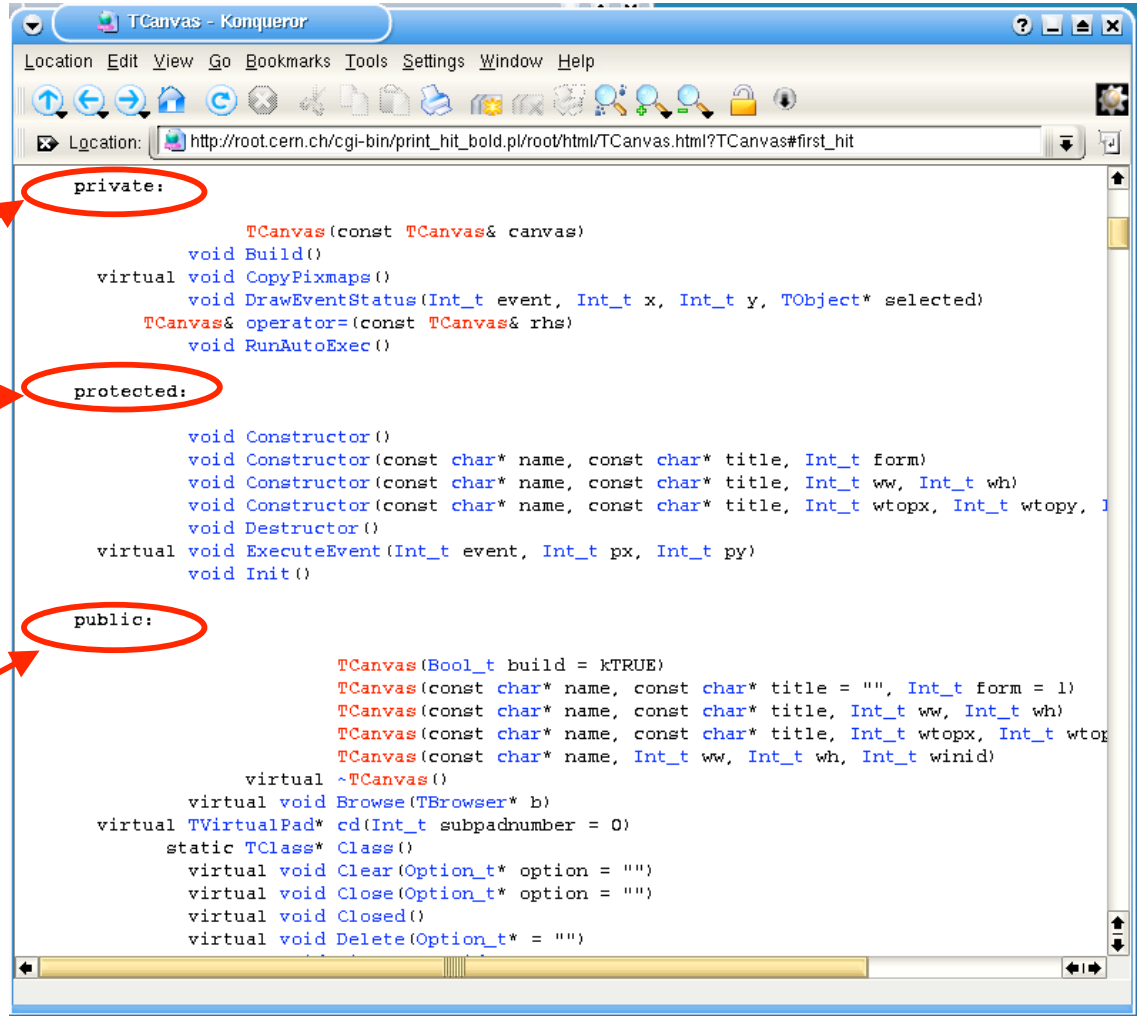


La liste des méthodes d'une classe

- Il existe trois types de méthodes: "private", "protected", "public"

on ne peut utiliser ni les méthodes "private" ni "protected"...

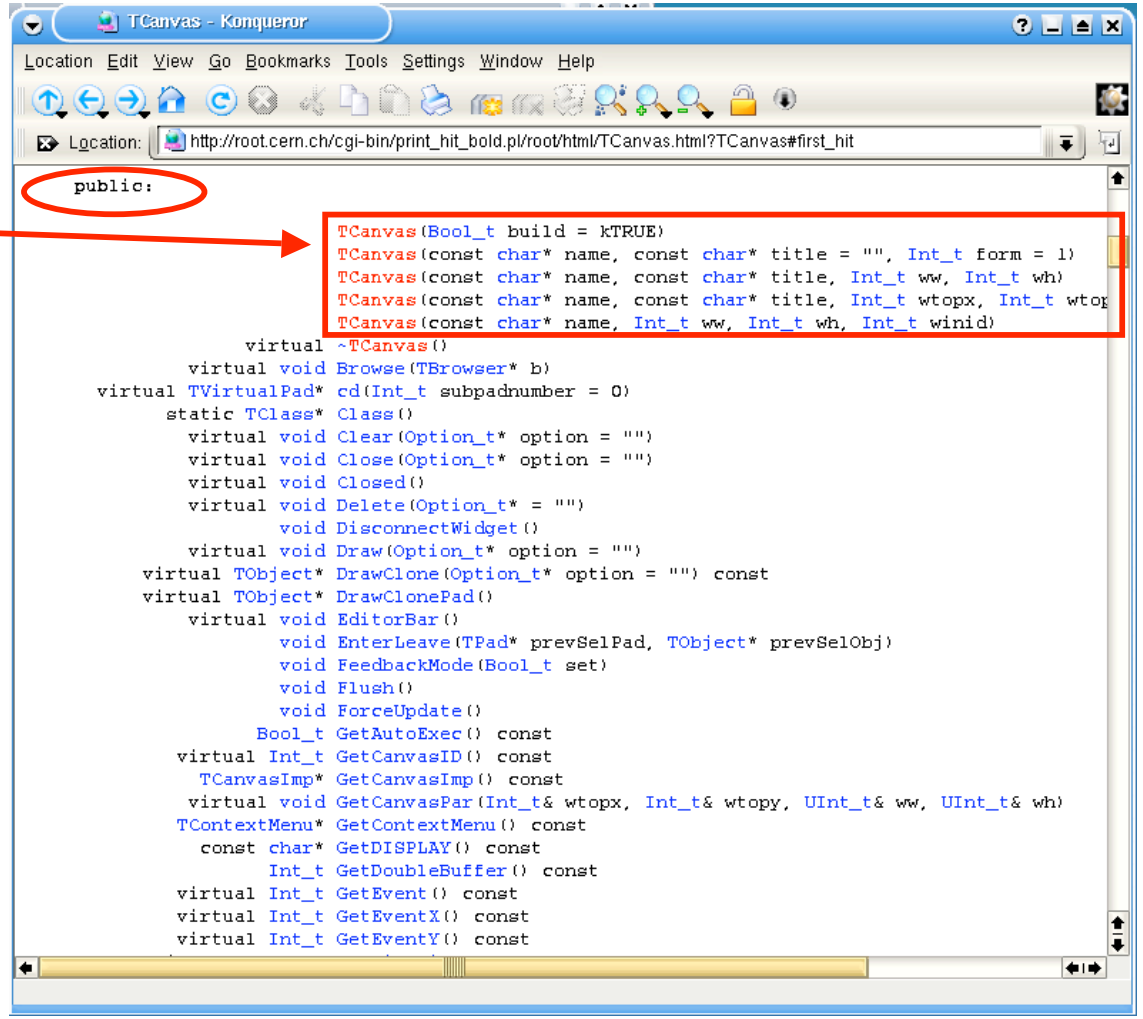
... donc on ne regardera que les méthodes "public"



La liste des méthodes d'une classe

- La liste des méthodes "public" est toujours organisée de la même façon:

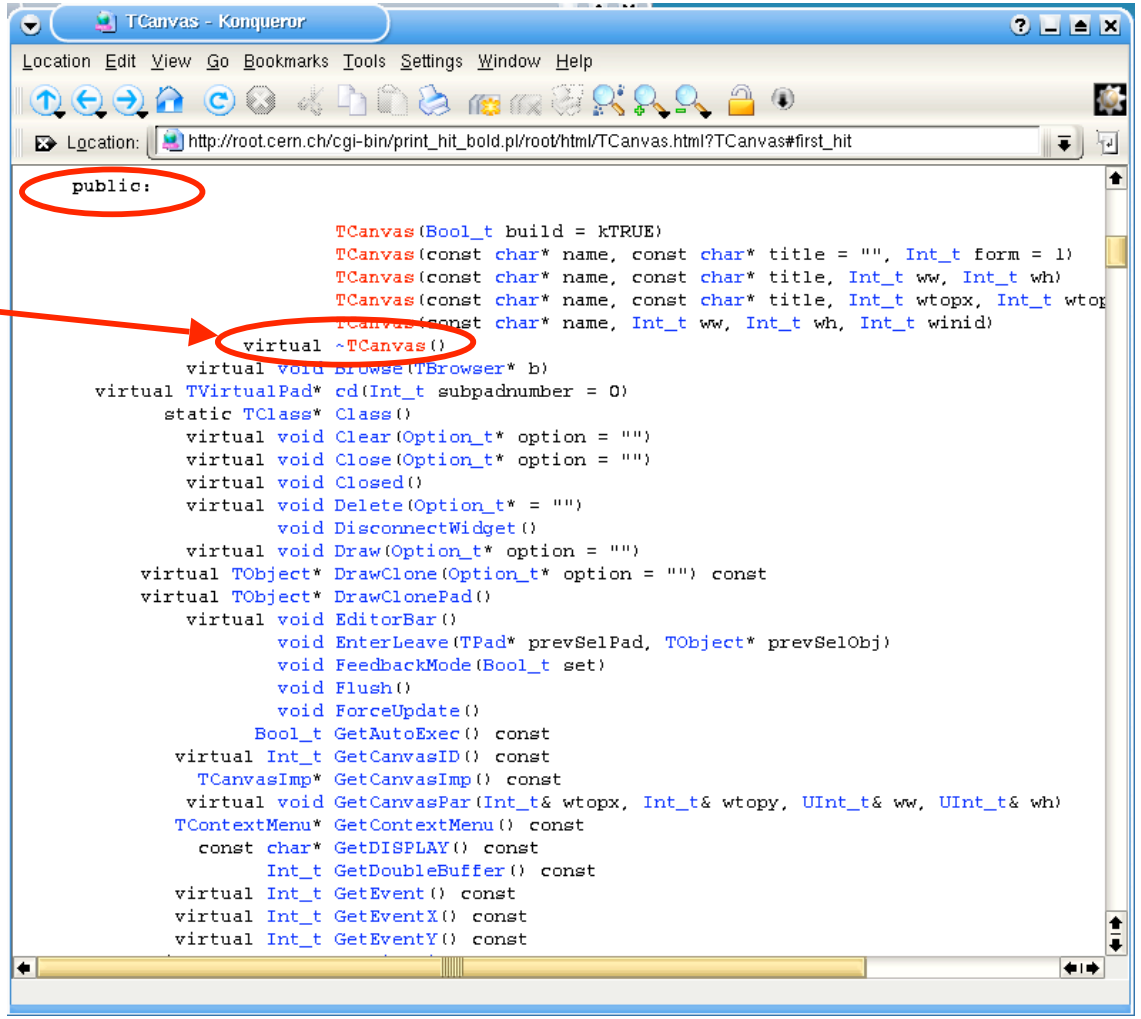
D'abord, on a les *constructeurs* de la classe (méthodes avec le même nom que la classe)



La liste des méthodes d'une classe

- La liste des méthodes "public" est toujours organisée de la même façon:

Une méthode *destructeur* "~TypeObjet()" appelée quand on détruit l'objet, e.g. avec delete*



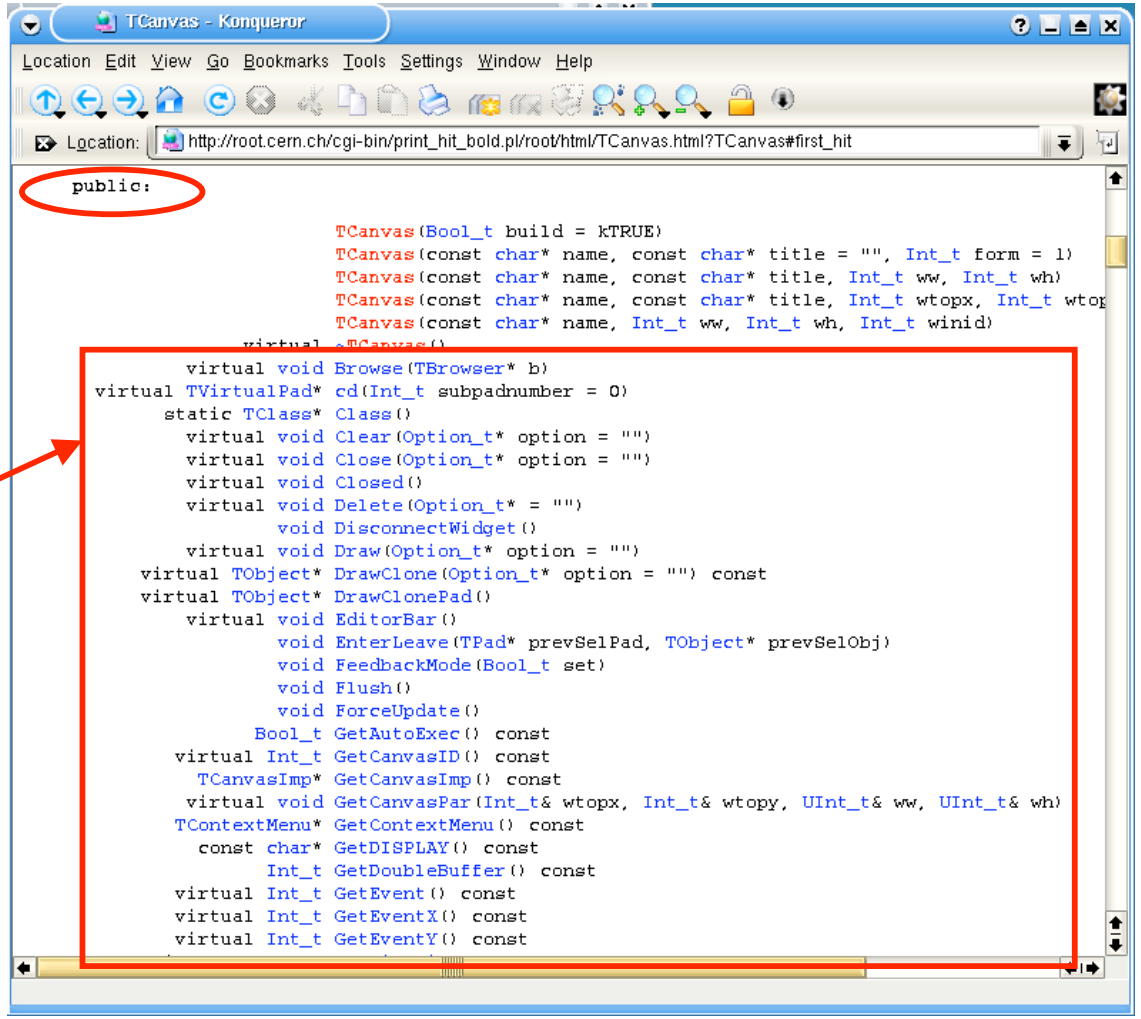
*On ne peut pas utiliser cette méthode directement

Trouver toutes les méthodes d'une classe

- La liste des méthodes "public" est toujours organisée de la même façon:

La liste alphabétique de toutes les méthodes de la classe…?

Où est la méthode "Divide"?

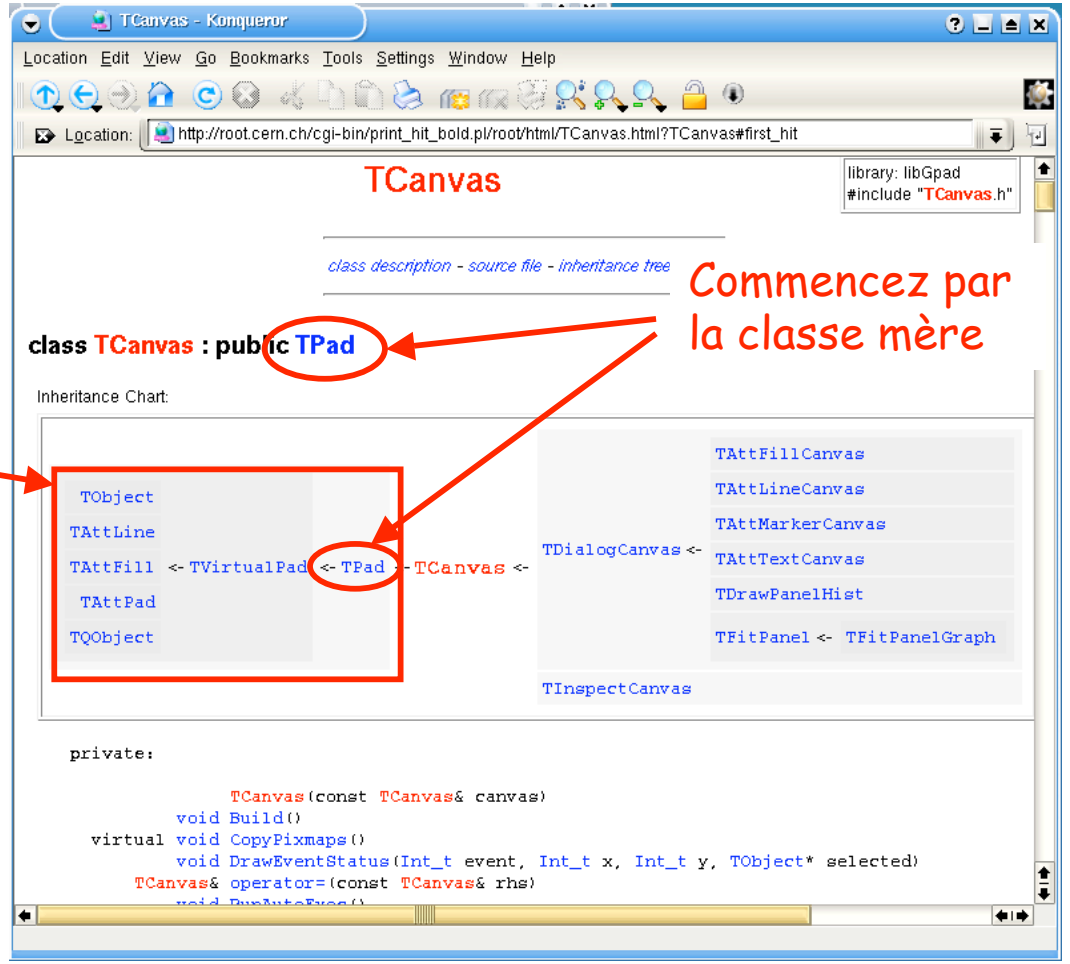


Trouver toutes les méthodes d'une classe

- Si une classe semble manquer une méthode, elle est peut-être définie par ses (grand-)parents

ASTUCE:
Les objets d'une classe profitent de tout le savoir-faire de leurs aïeuls…

…les méthodes passent de mère en fille!!

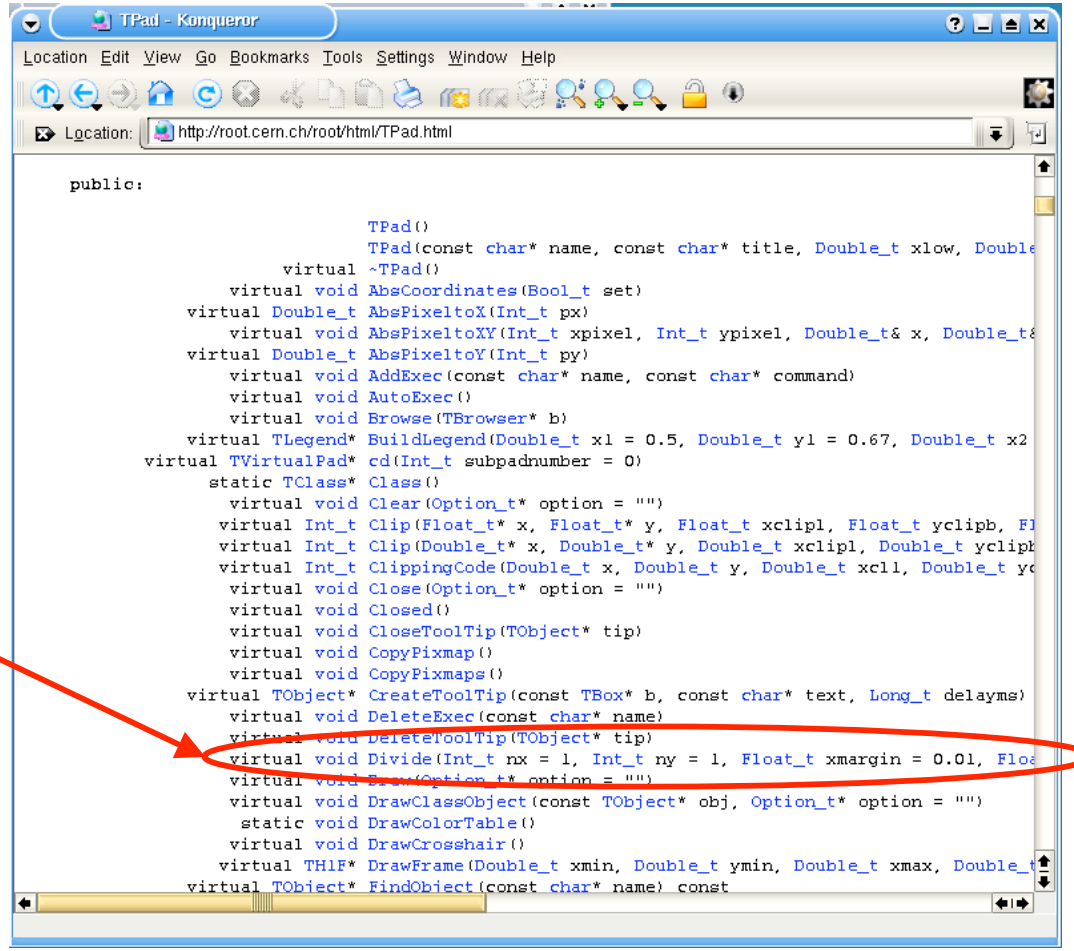


Trouver toutes les méthodes d'une classe

- Si une classe semble manquer une méthode, elle est peut-être définie par ses (grand-)parents

Trouvée!
Regardons la doc…*

*…on expliquera la déclaration (types de variables, arguments par défaut etc.) tout à l'heure

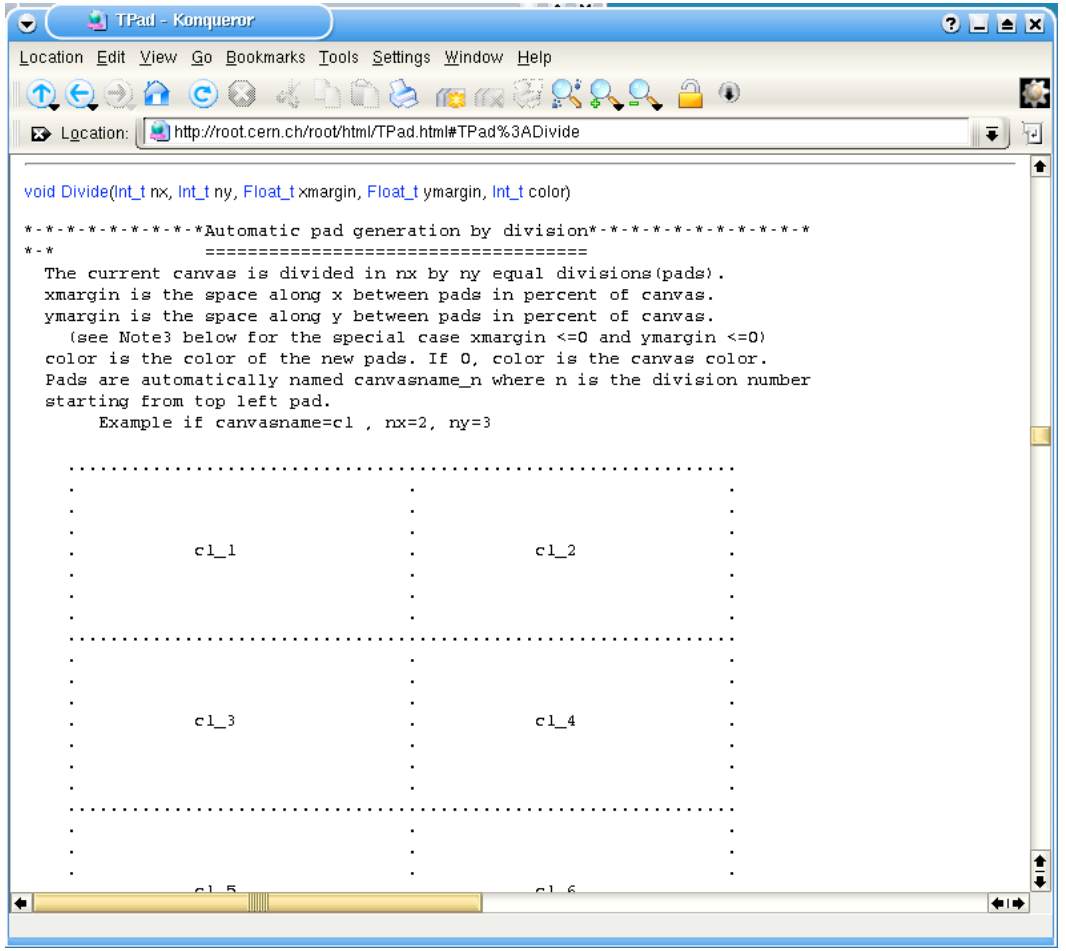


Trouver toutes les méthodes d'une classe

- Si une classe semble manquer une méthode, elle est peut-être définie par ses (grand-)parents

Voici la doc en ligne de la méthode "Divide" de la classe "TPad", qui est aussi une méthode de la classe "TCanvas", classe fille de "TPad".

Ouf!



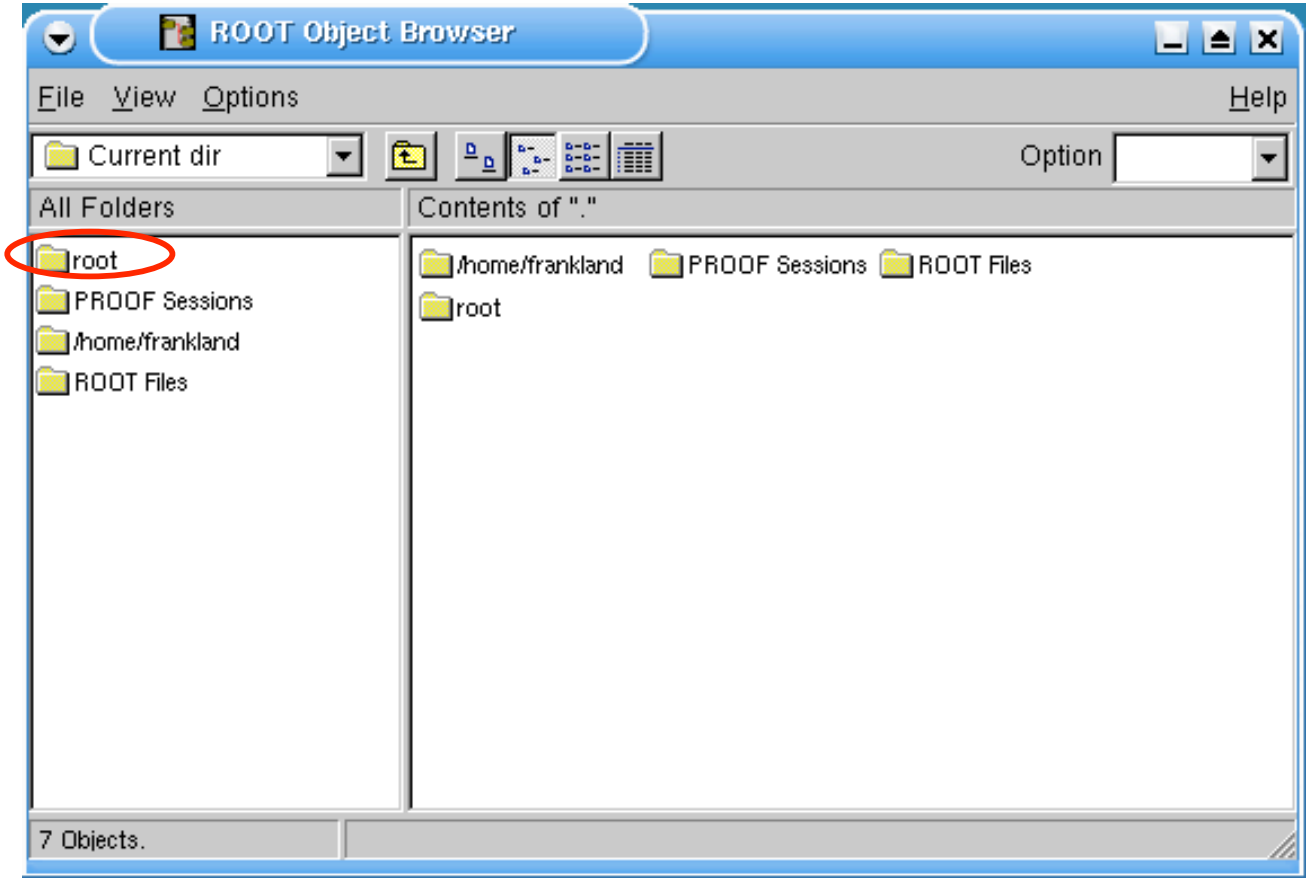
A la recherche des objets perdus

Ou: pourquoi les objets ont des noms

Retrouver un objet perdu

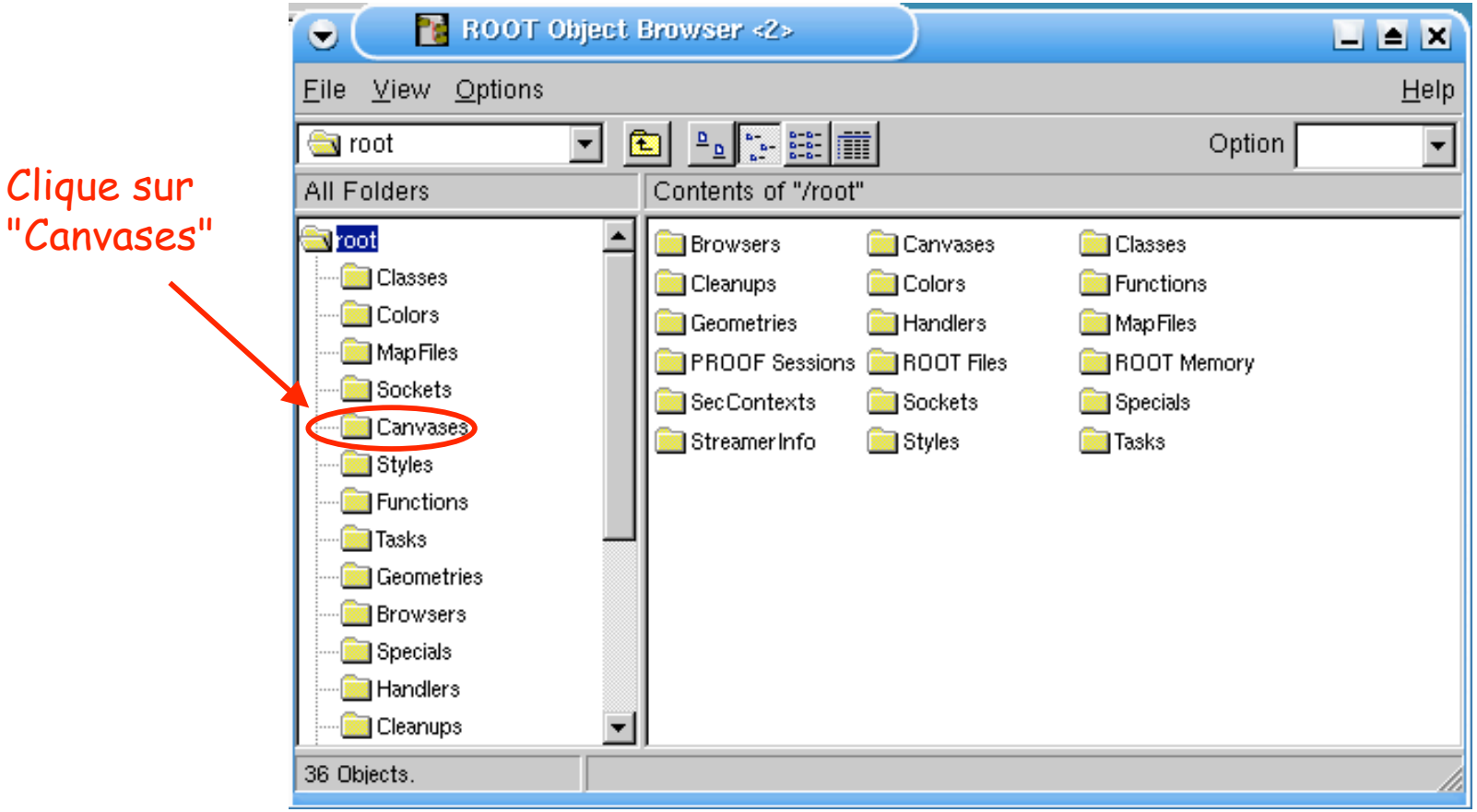
- ROOT tient à jour des listes d'objets, ce qui permet de les retrouver facilement

Double-clique sur "root"



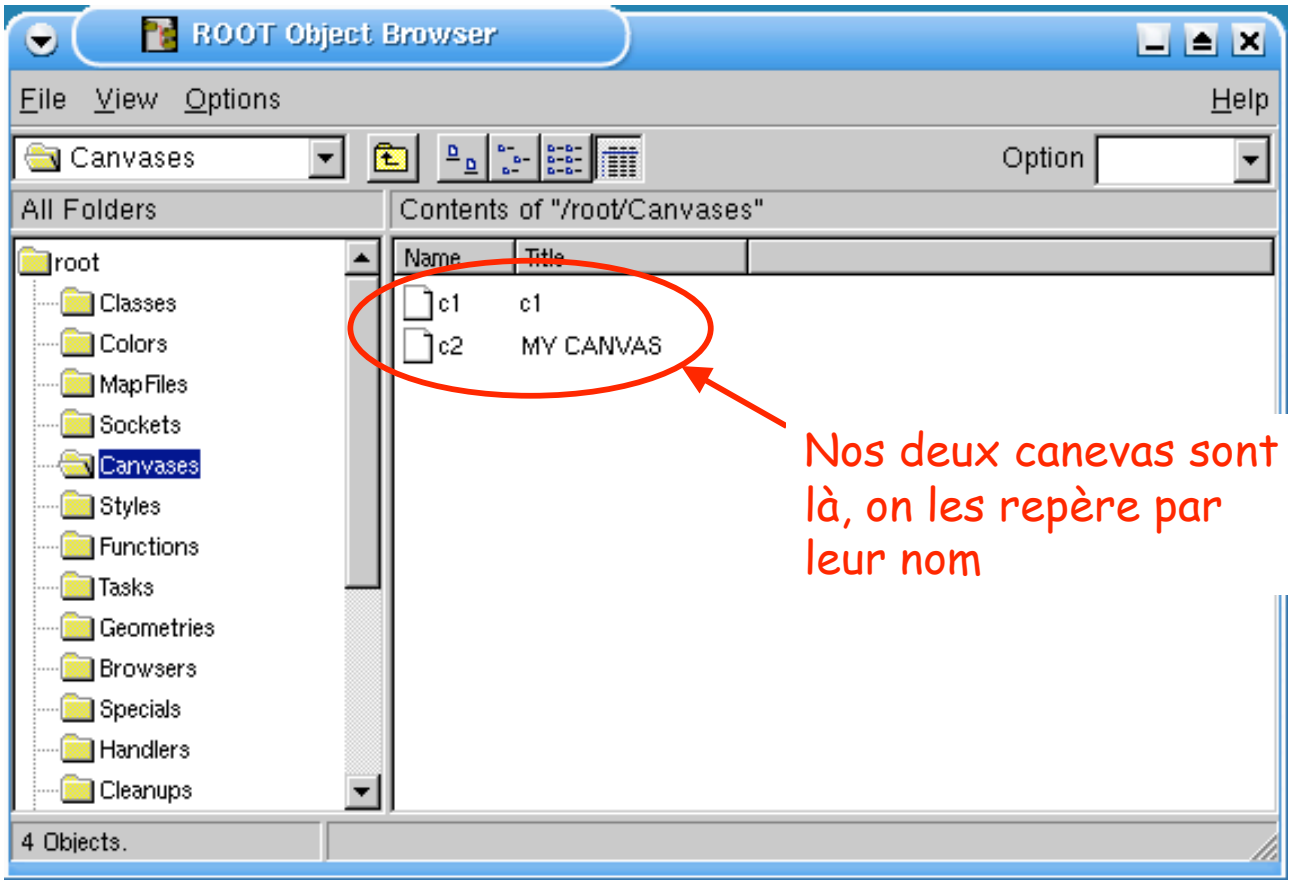
Retrouver un objet perdu

- On peut naviguer dans ces listes en utilisant le TBrowser



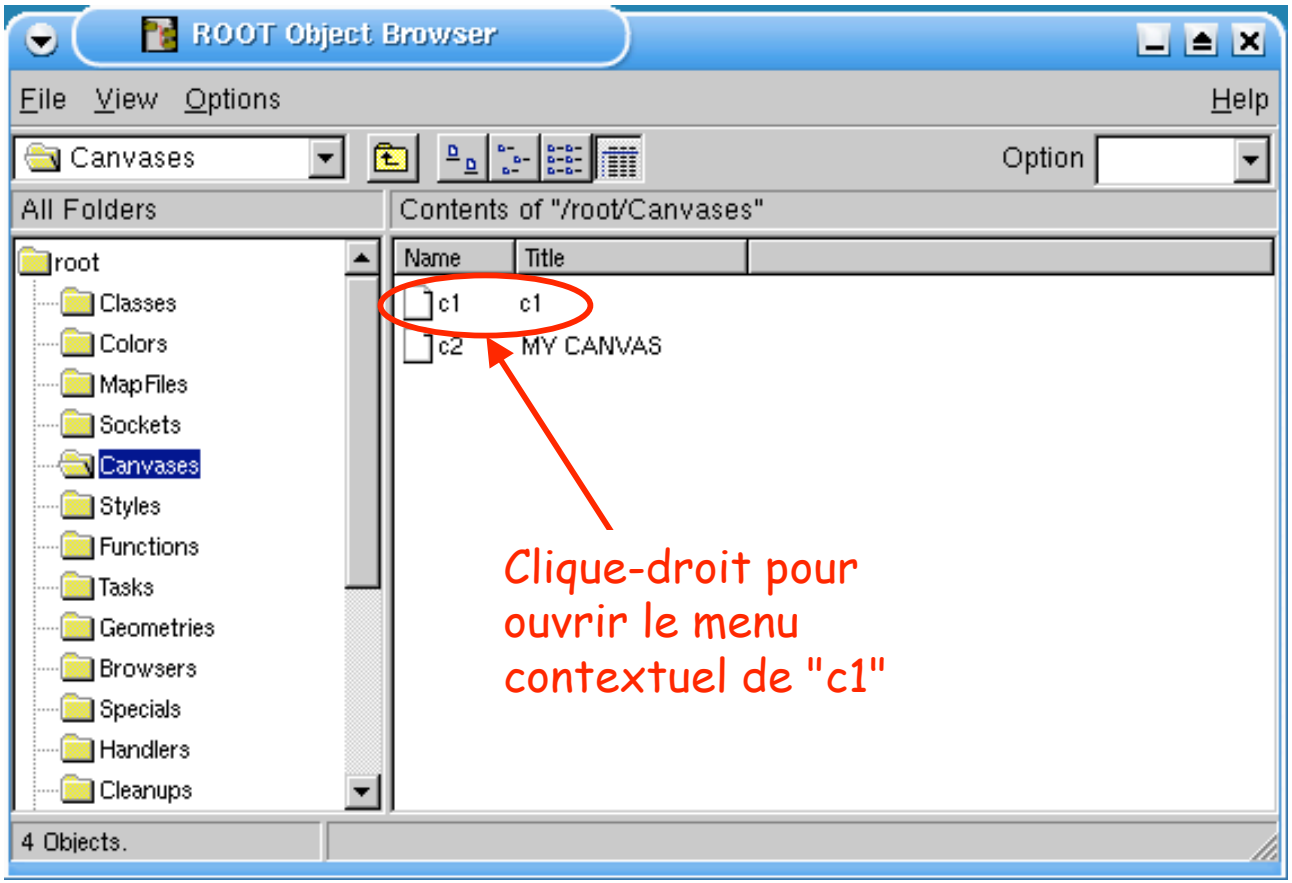
Retrouver un objet perdu

- Chaque objet doit porter un nom unique, pour qu'on puisse le retrouver



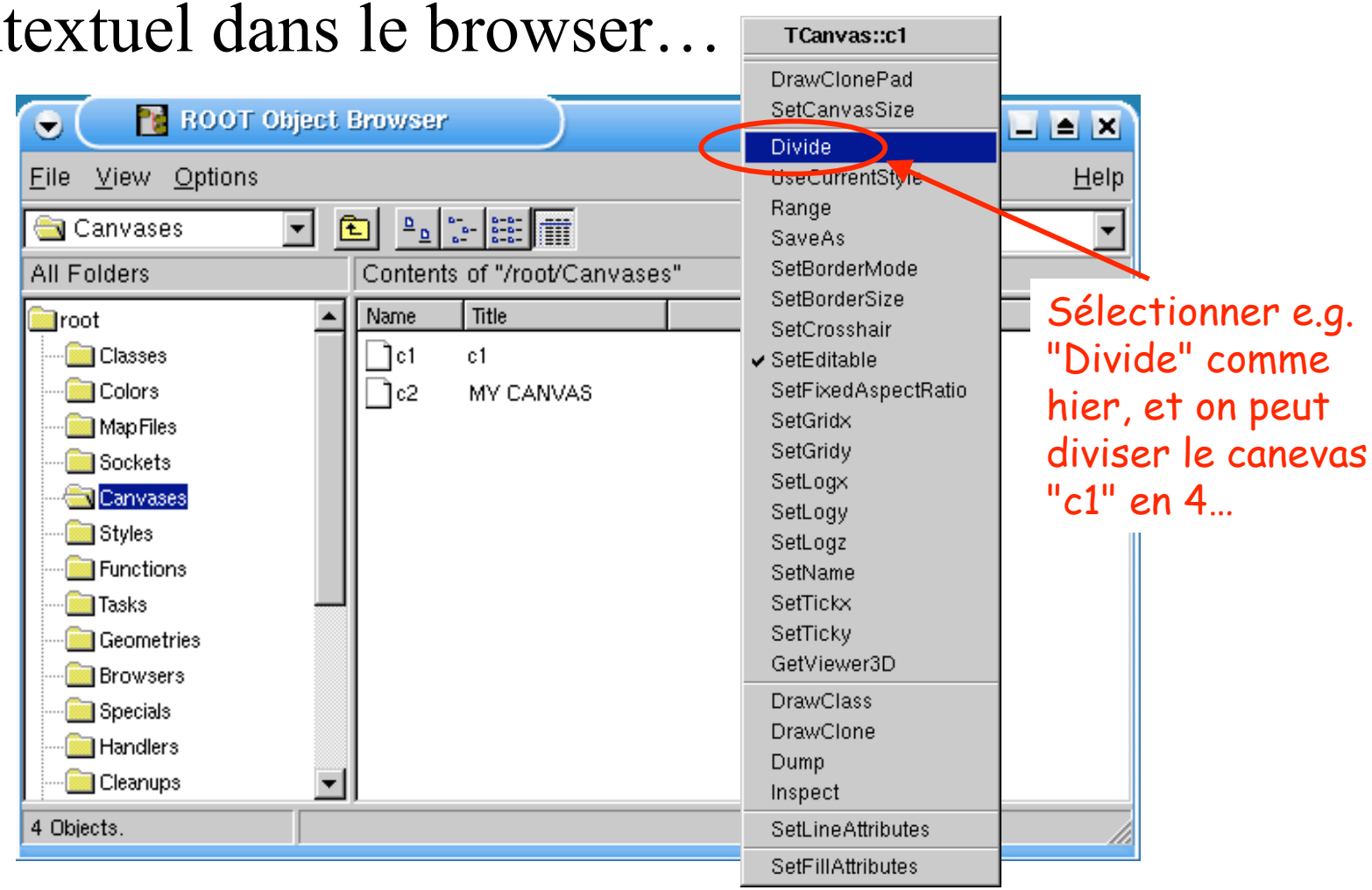
Retrouver un objet perdu

- On peut agir sur un objet à travers son menu contextuel dans le browser…



Retrouver un objet perdu

- On peut agir sur un objet à travers son menu contextuel dans le browser…



Retrouver un objet perdu

- …ou en récupérant son adresse, on agira avec un pointeur d'objet:

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

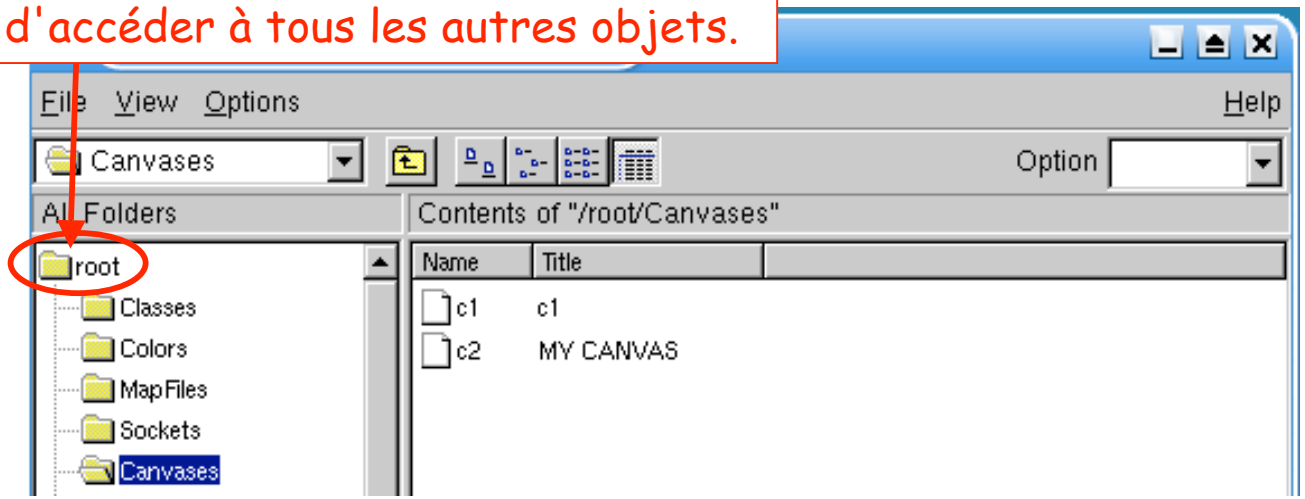
Retrouver un objet perdu

- …ou en récupérant son adresse, on agira avec un pointeur d'objet:

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

Ce pointeur d'objet contient l'adresse du dossier racine "root" dans le TBrowser.

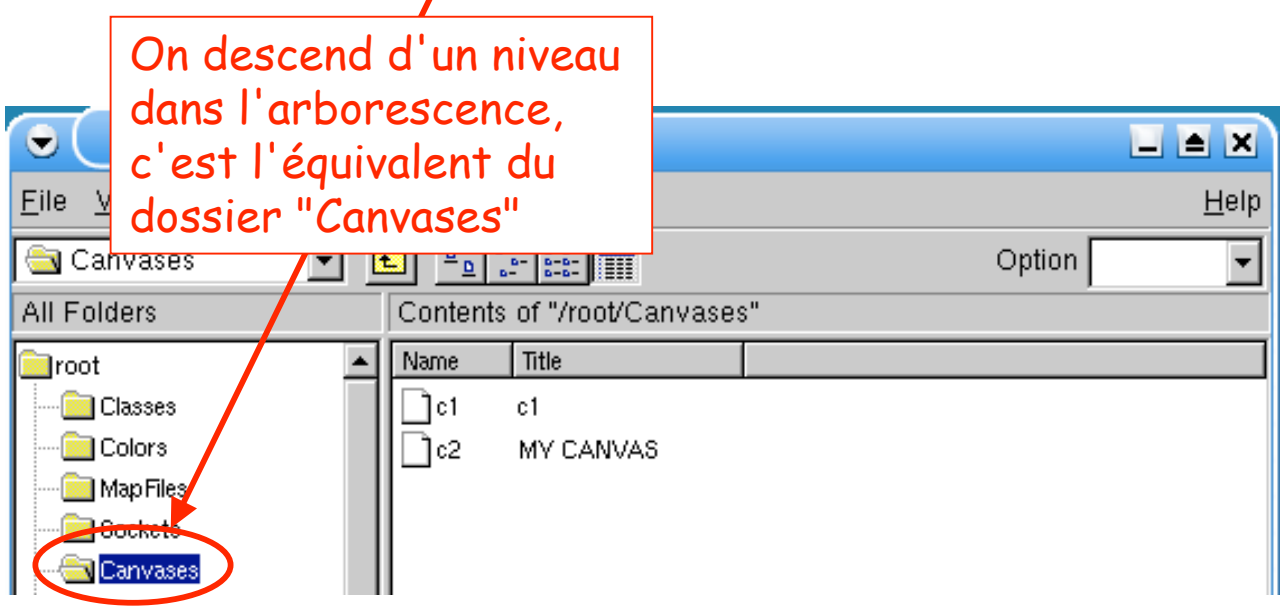
Il permet d'accéder à tous les autres objets.



Retrouver un objet perdu

- …ou en récupérant son adresse, on agira avec un pointeur d'objet:

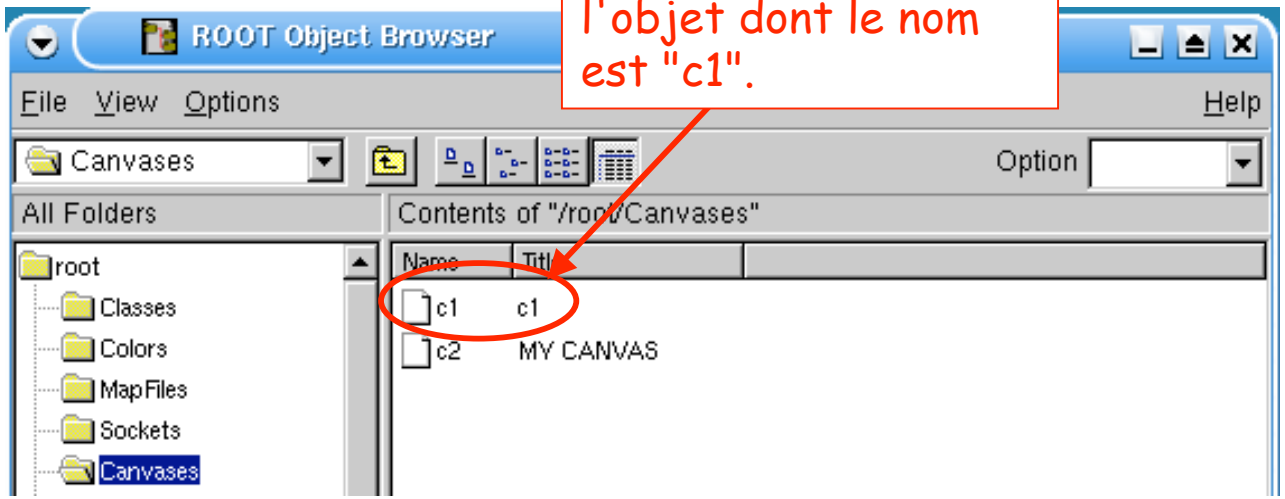
```
gROOT->GetListOfCanvases()->FindObject("c1");
```



Retrouver un objet perdu

- …ou en récupérant son adresse, on agira avec un pointeur d'objet:

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



Retrouver un objet perdu

- Si l'on n'est pas sûr dans quel dossier il faut chercher, on peut faire une recherche récursive dans tous les dossiers:

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

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

C'est la formule magique qui permet de retrouver à peu près n'importe quel objet à n'importe quel moment.

On l'utilisera tout le temps!!

Retrouver un objet perdu

- Ensuite il n'y a plus qu'à mettre l'adresse dans un pointeur approprié et l'utiliser:

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

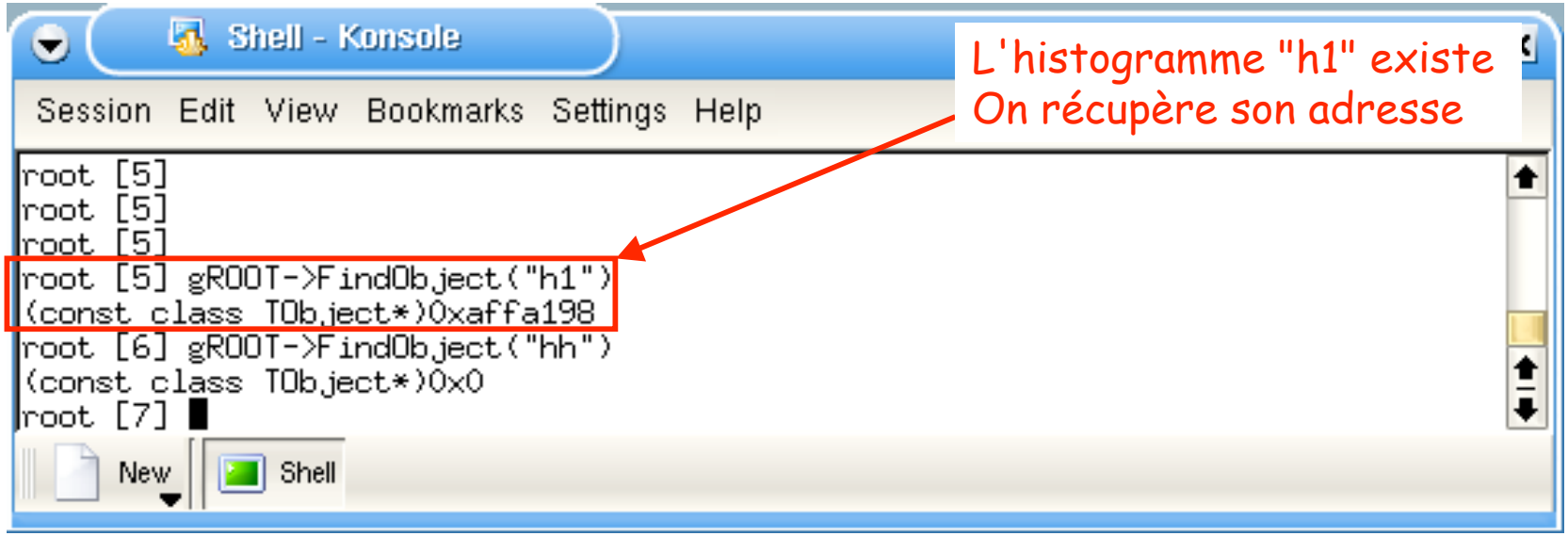
On doit repréciser le type de l'objet recherché ici

Exemple: on cherche le canevas "c1" et on efface son contenu:

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

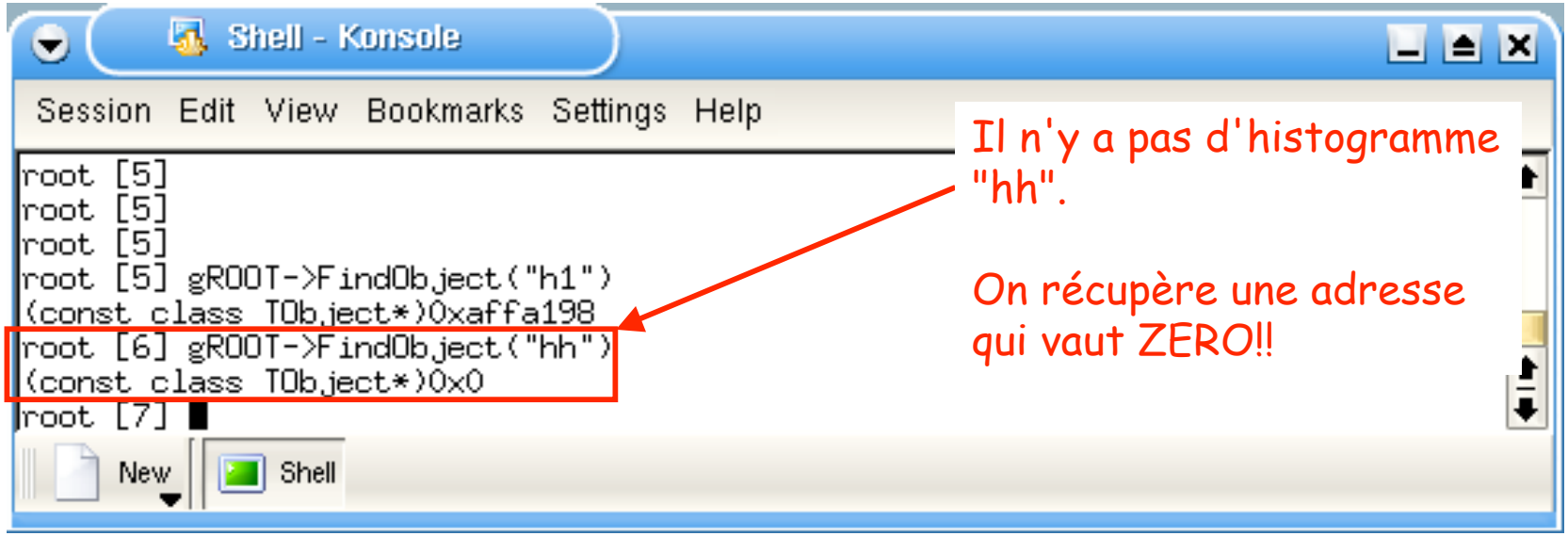
Savoir si un objet existe

- On utilisera la même fonction quand on voudra savoir si tel ou tel objet a déjà été créé



Savoir si un objet existe

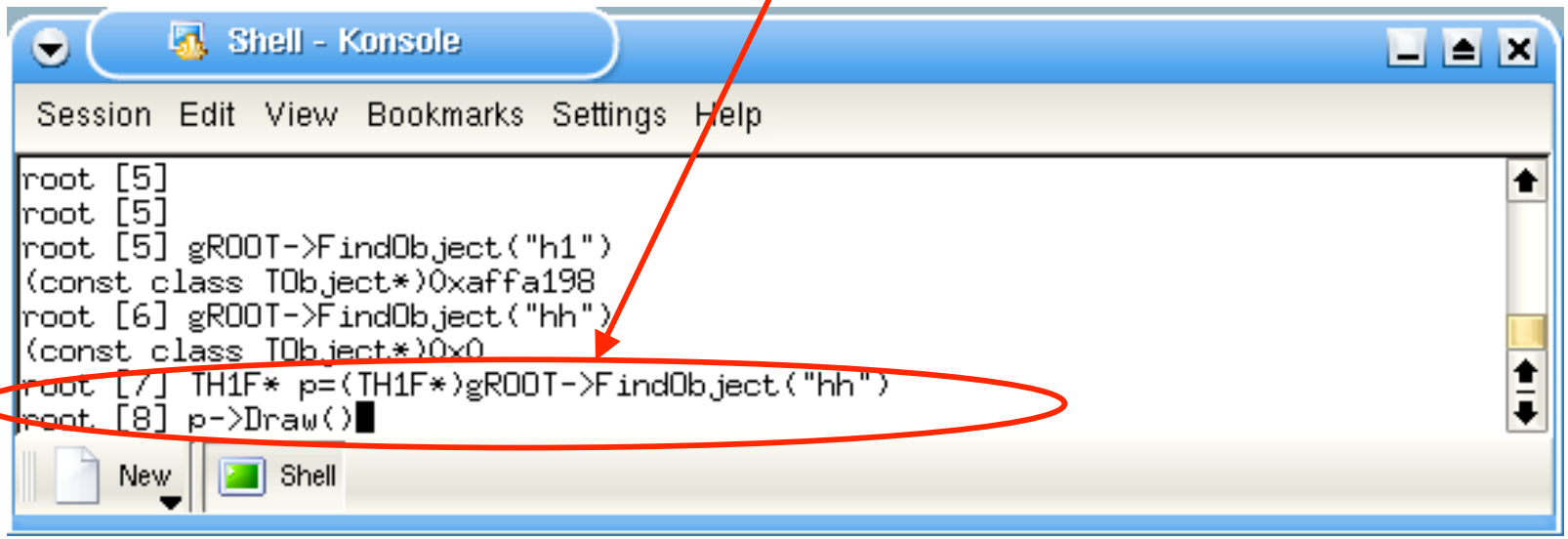
- On utilisera la même fonction quand on voudra savoir si tel ou tel objet a déjà été créé



Savoir si un objet existe

- En toute rigueur, il faut toujours tester la valeur d'un pointeur pour être sûr que l'adresse est valable…

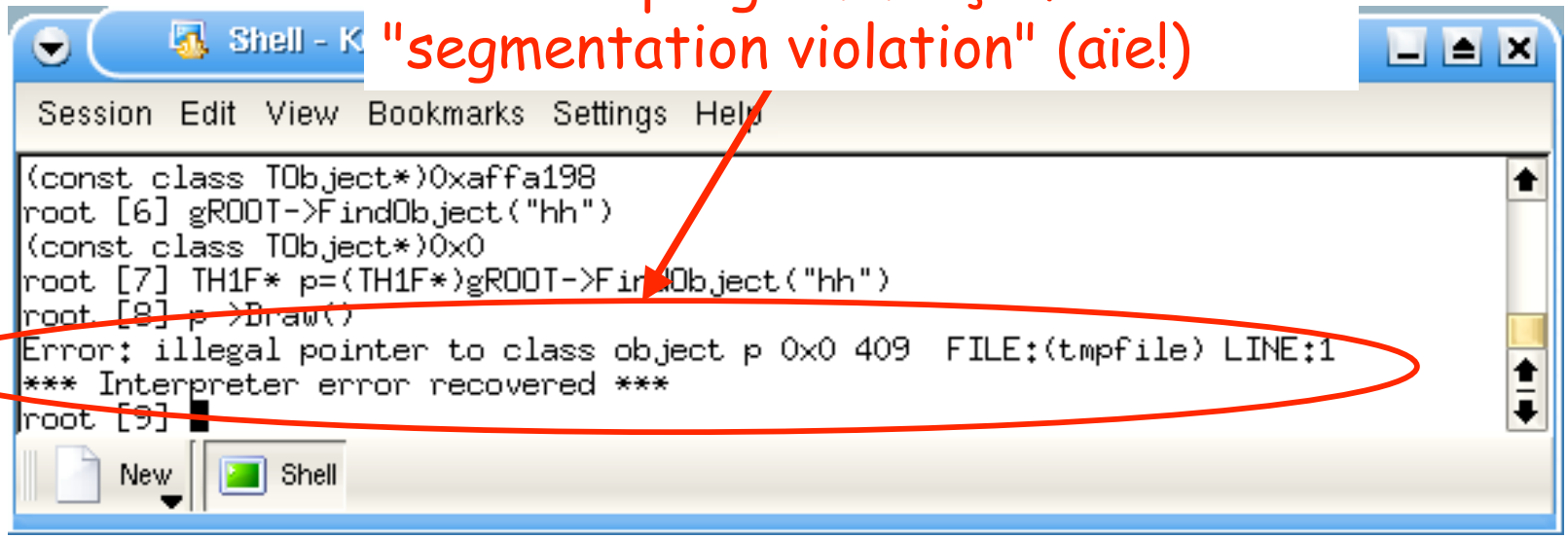
Qu'est-ce qui se passe quand on essaie d'utiliser un pointeur nul ?



Savoir si un objet existe

- En toute rigueur, il faut toujours tester la valeur d'un pointeur pour être sûr que l'adresse est valable…

L'interpréteur est gentil…
…dans un programme ça ferait
"segmentation violation" (aïe!)

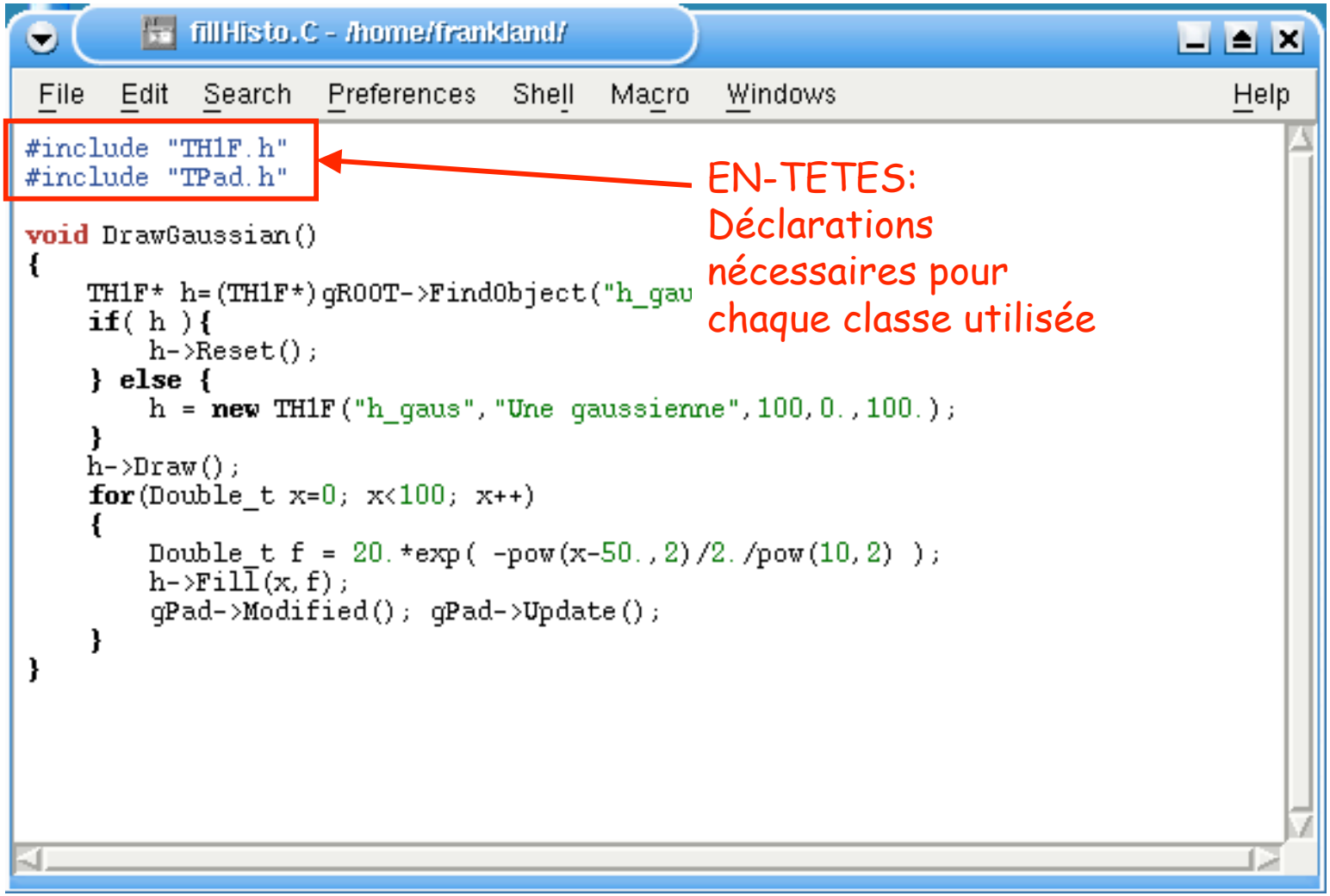


Ecriture de fonctions

La programmation C++ par les nuls

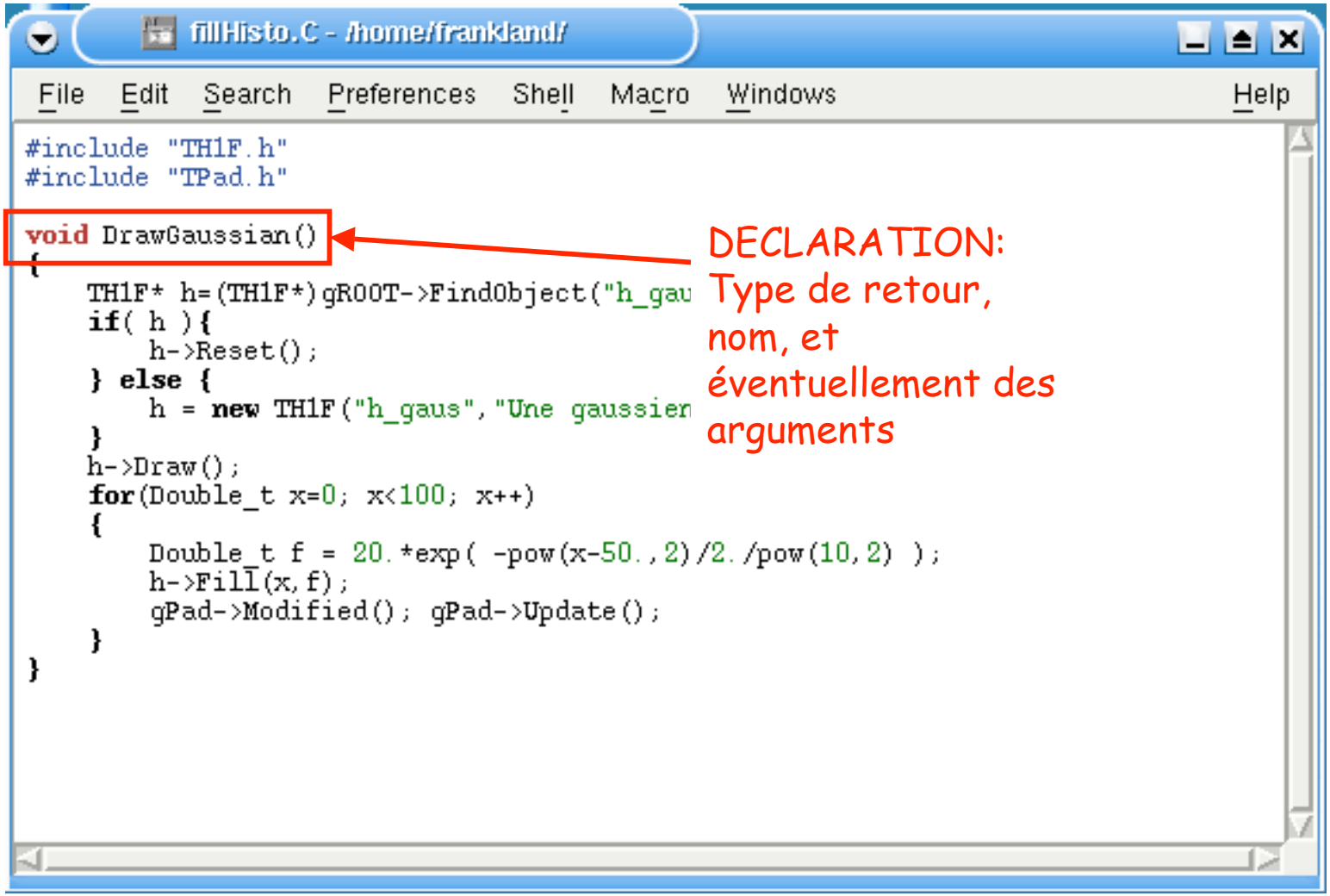
Une fonction

- Voici un exemple d'une fonction C++



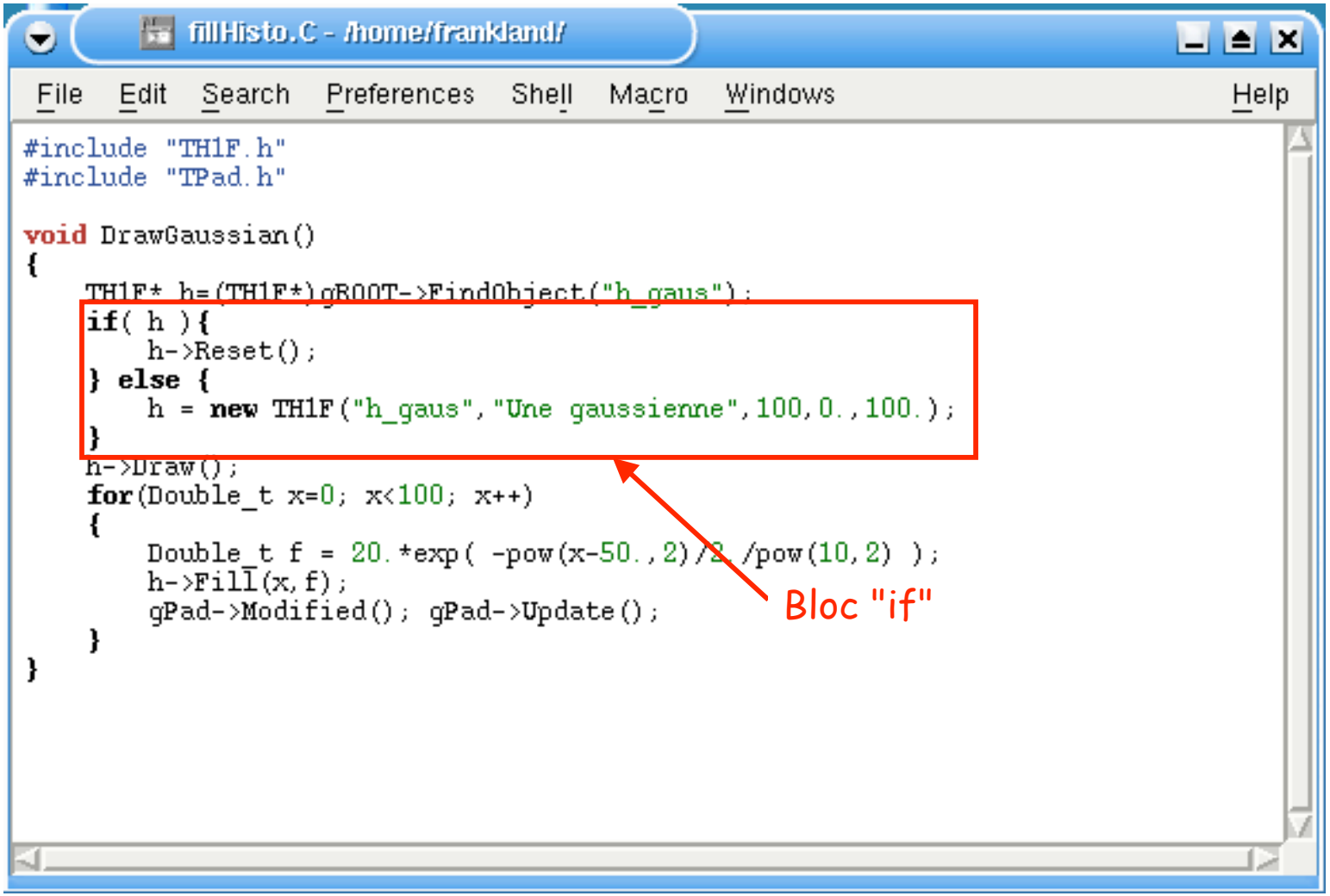
Une fonction

- Voici un exemple d'une fonction C++



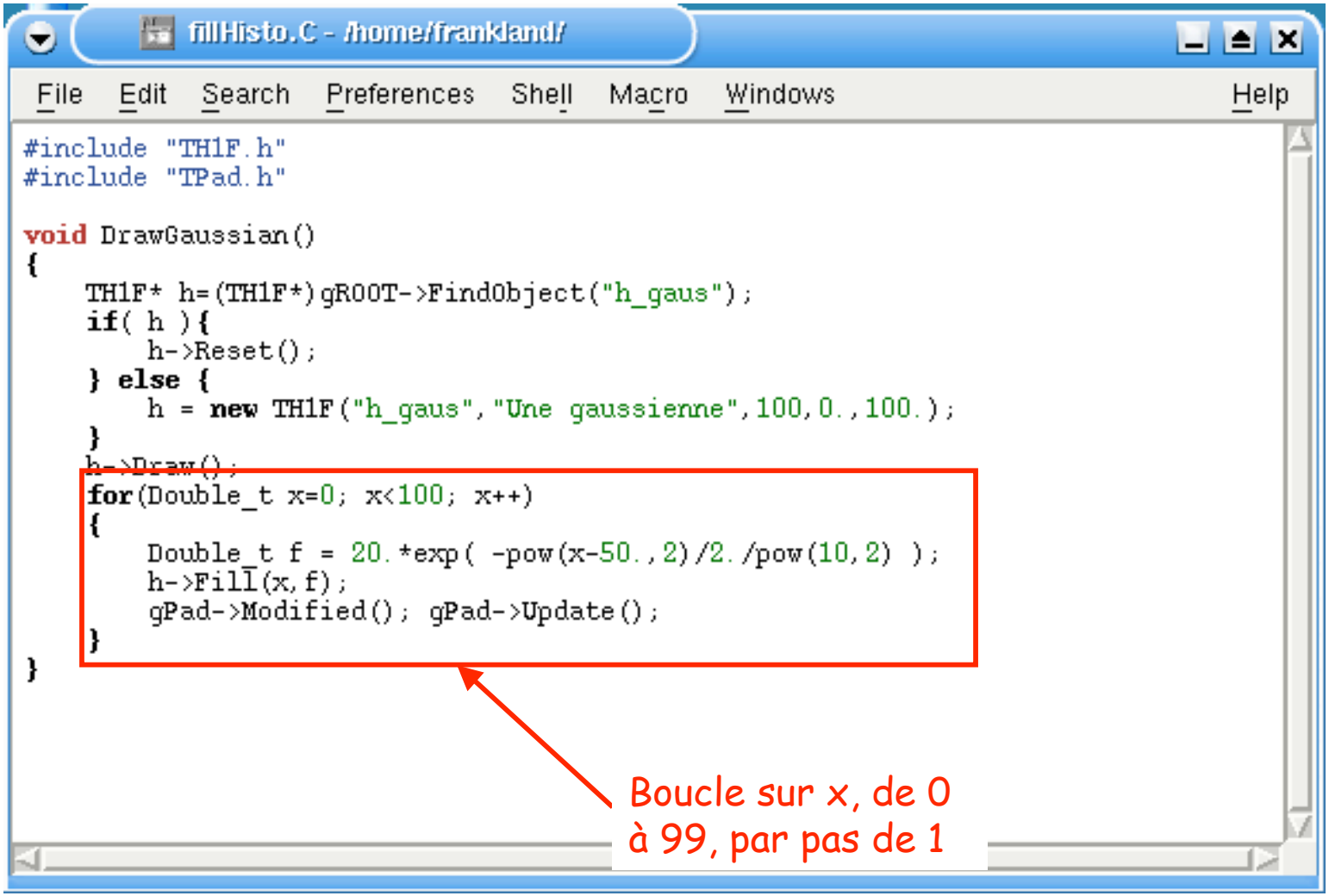
Une fonction

- Voici un exemple d'une fonction C++



Une fonction

- Voici un exemple d'une fonction C++



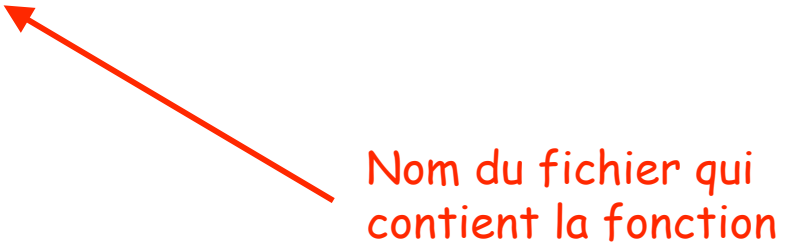
Une fonction

- Voici un exemple d'une fonction C++

Compilation et utilisation

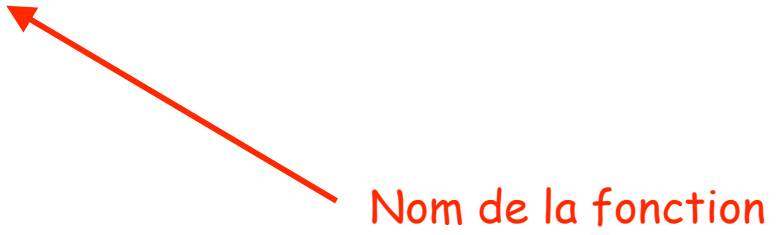
- On compile et on charge la fonction:

```
.L fillHisto.C+
```



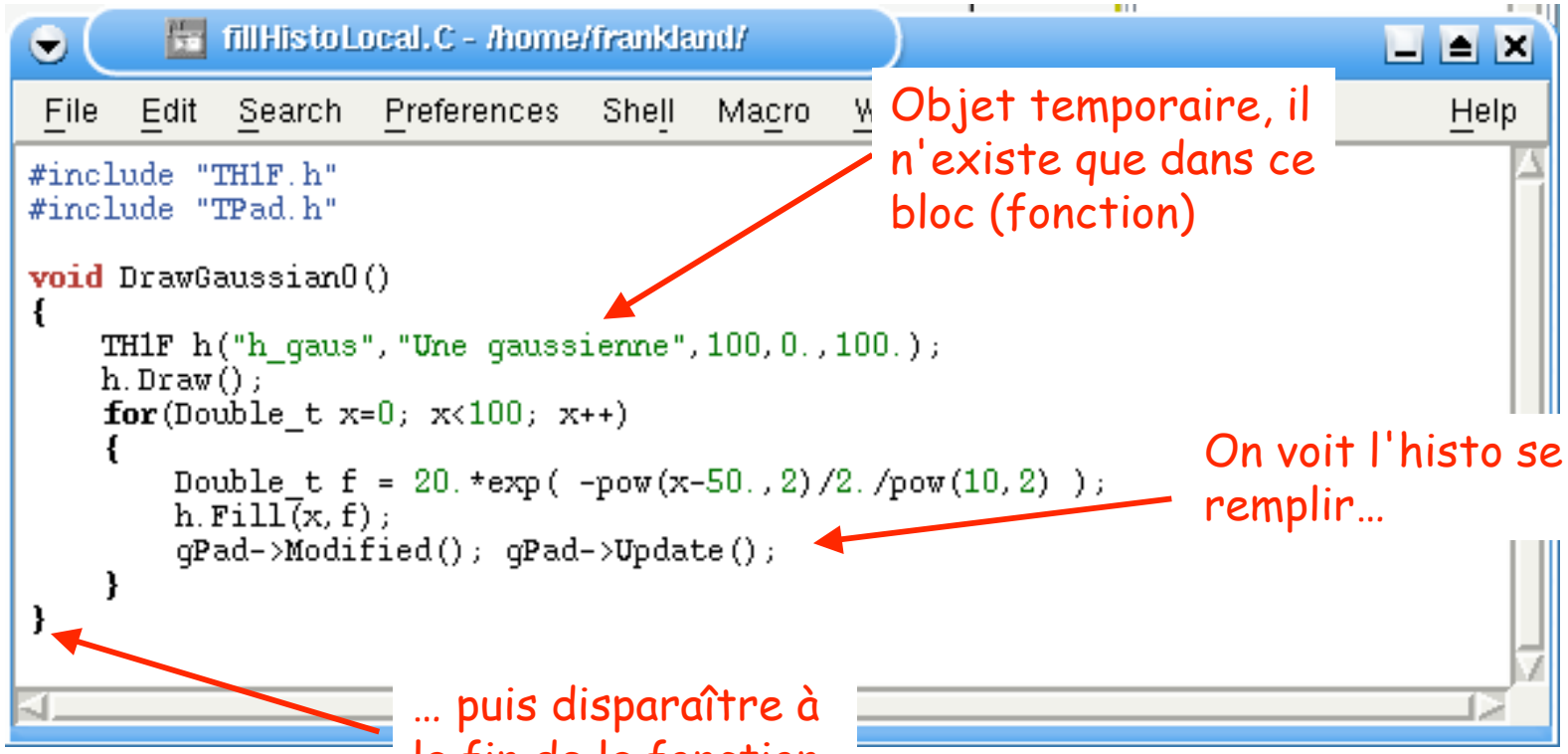
- On exécute la fonction:

```
DrawGaussian()
```



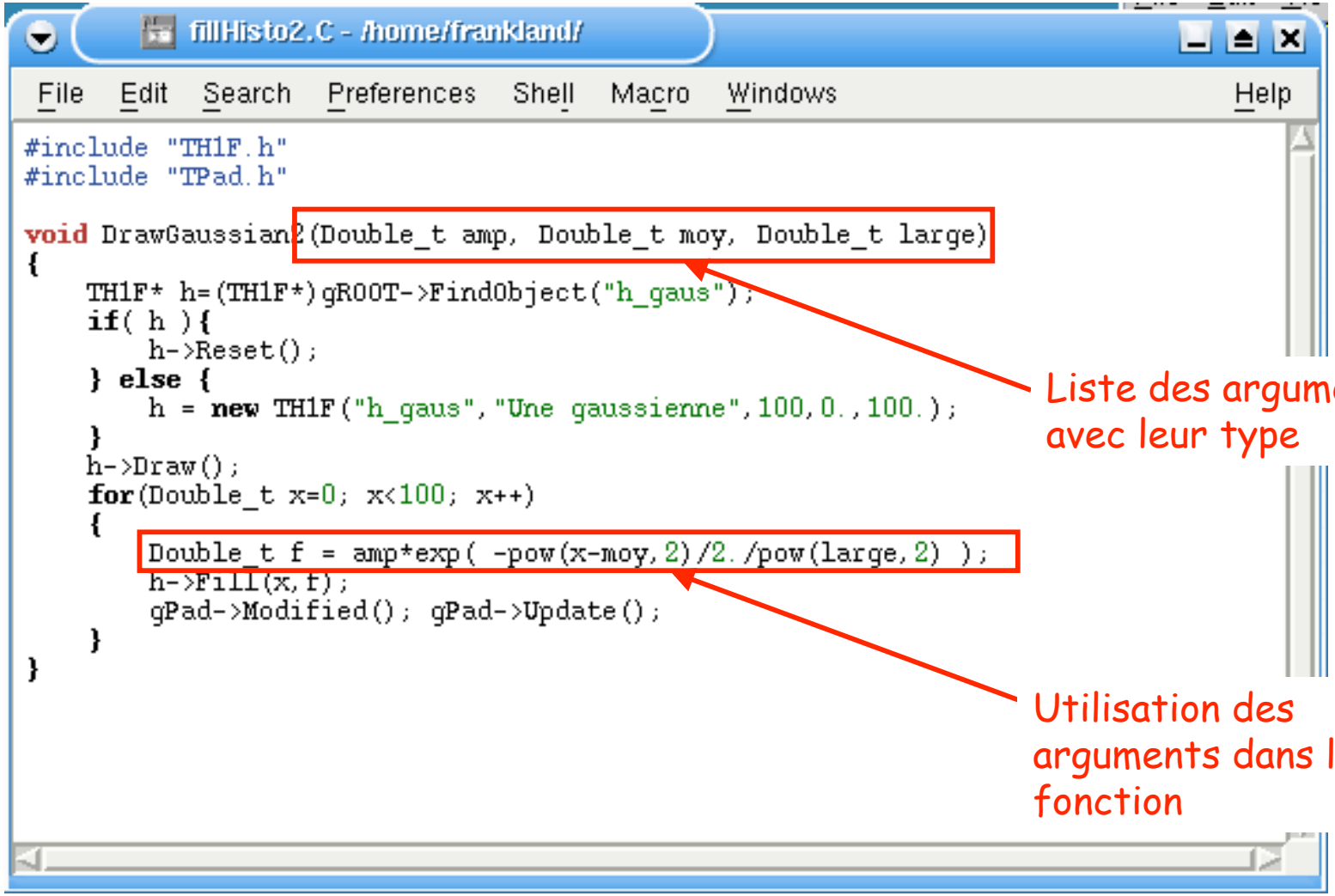
Pourquoi utiliser "new" pour créer l'histo ?

- Voyons ce qui se passe si on crée notre histogramme sans utiliser "new"



Comment utiliser les arguments

- Une fonction avec des arguments:



Valeurs d'arguments par défaut

fillHisto3.C - /home/frankland/

File Edit Search Preferences Shell Macro Windows Help

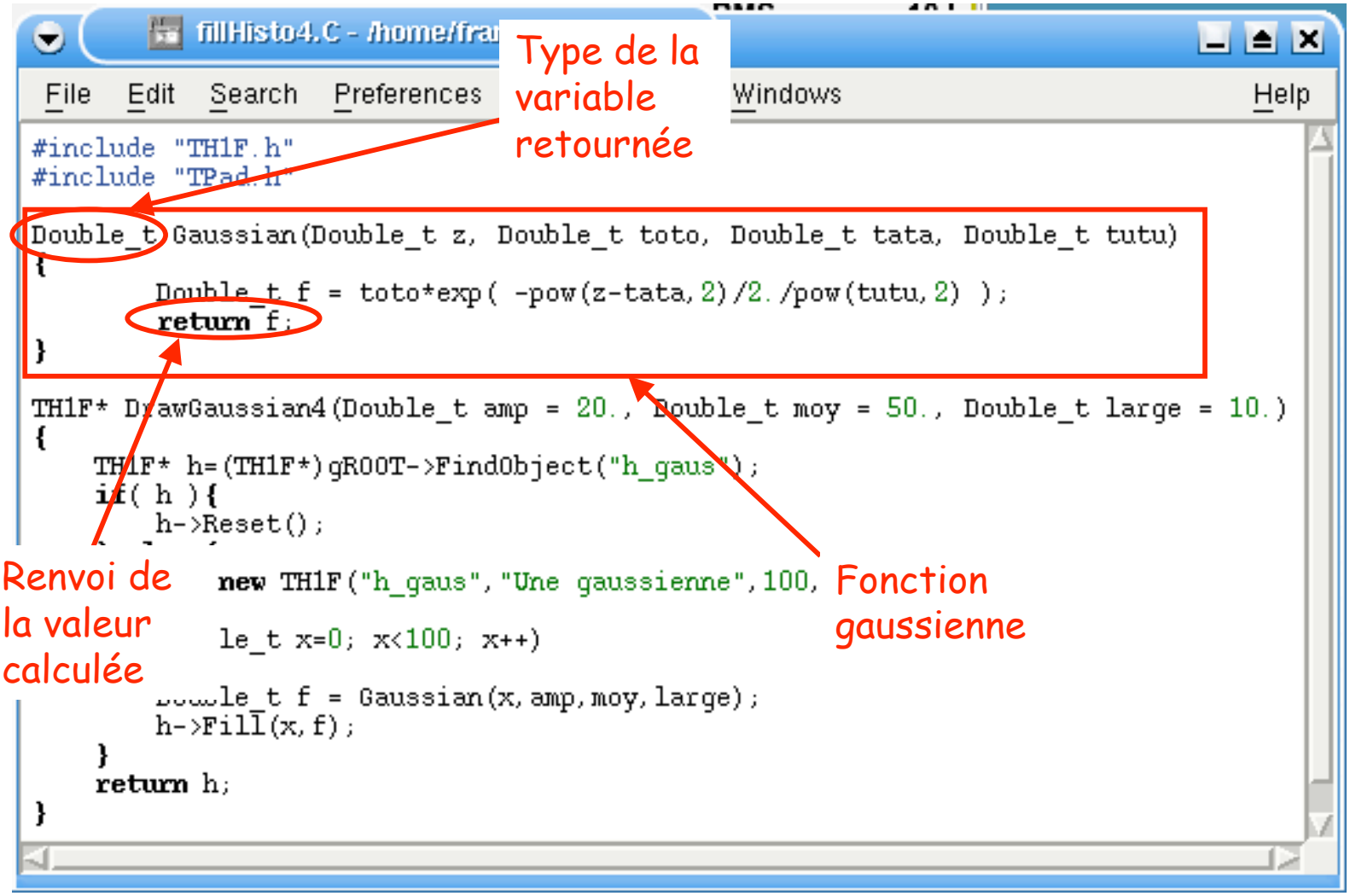
```
#include "TH1F.h"
#include "TPad.h"

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. Largeur par défaut

DrawGaussian() => amp=20, moy=50, large=10
DrawGaussian(5) => amp=5, moy=50, large=10
DrawGaussian(5,30) => amp=5, moy=30, large=10
DrawGaussian(5,30,5) => amp=5, moy=30, large=5

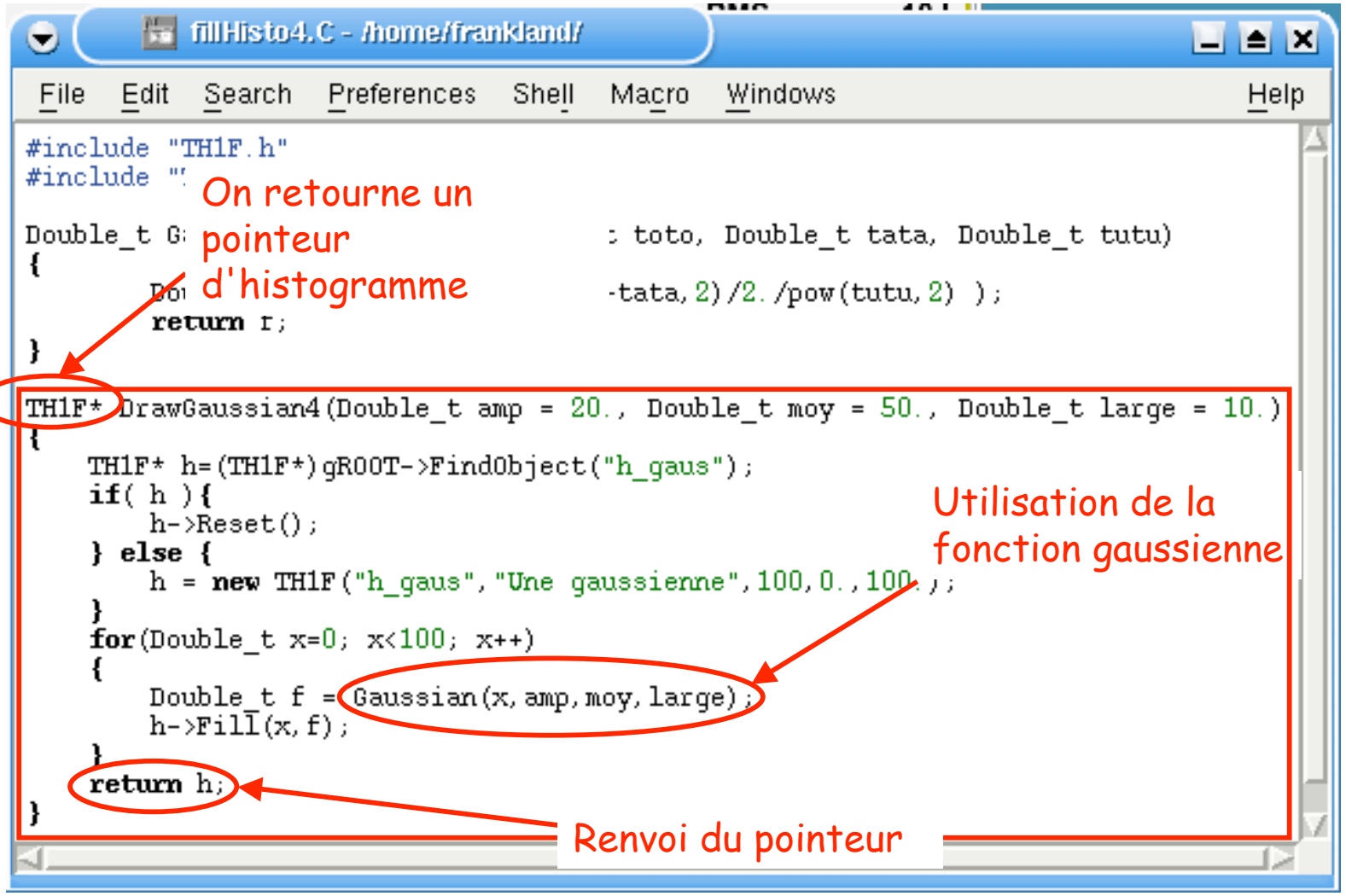
Retour de valeurs/objets

- Les fonctions peuvent retourner tous les types de variables



Retour de valeurs/objets

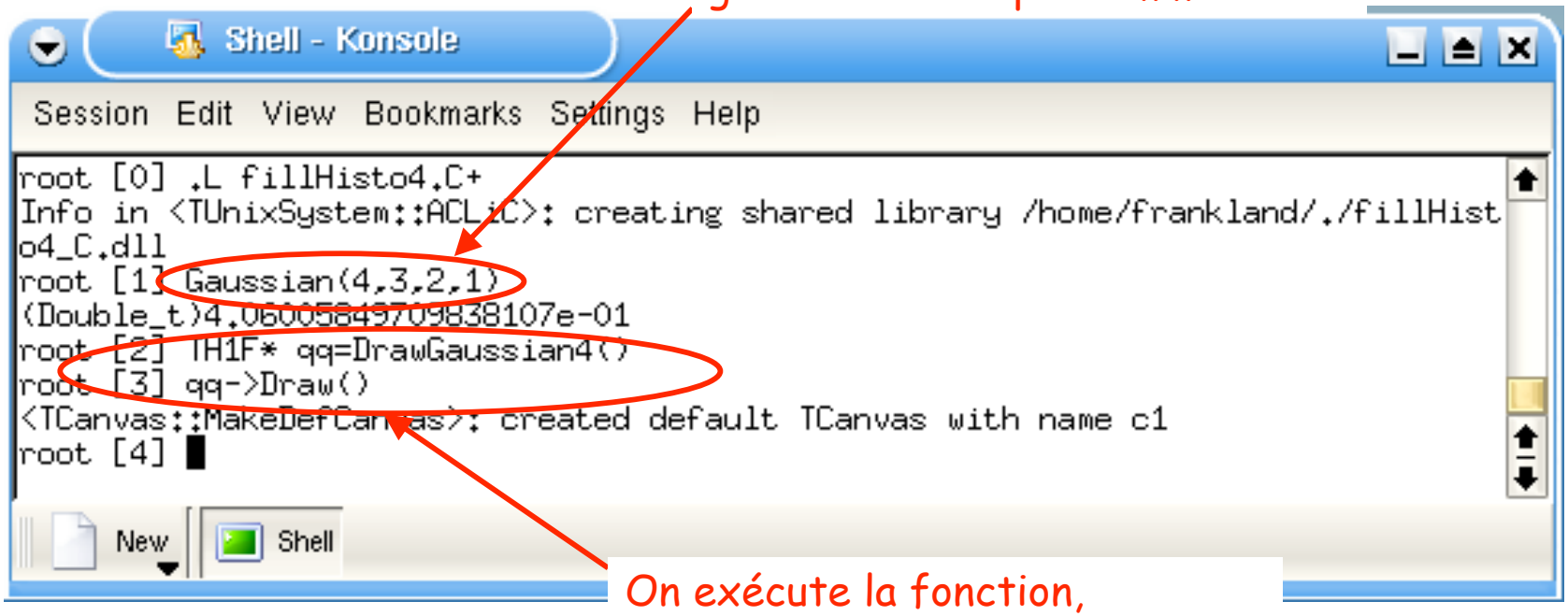
- Les fonctions peuvent retourner tous les types de variables




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

Double_t G:                          : toto, Double_t tata, Double_t tutu)
{
        Do                           -tata,2)/2./pow(tutu,2) );
        return r;
}

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;
}
```

Retour de valeurs/objets

- Utilisation de la fonction

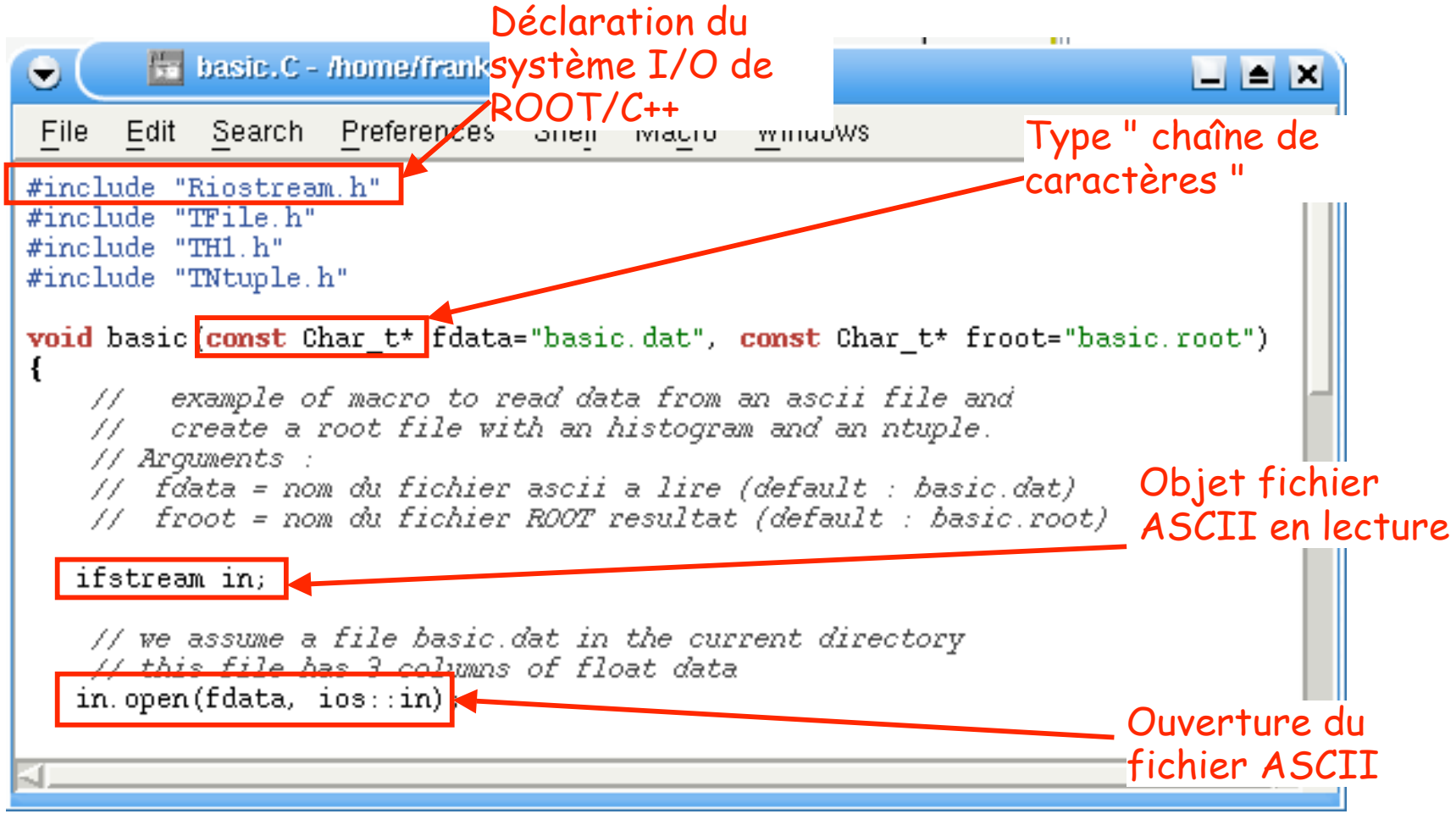


Exemple d'analyse

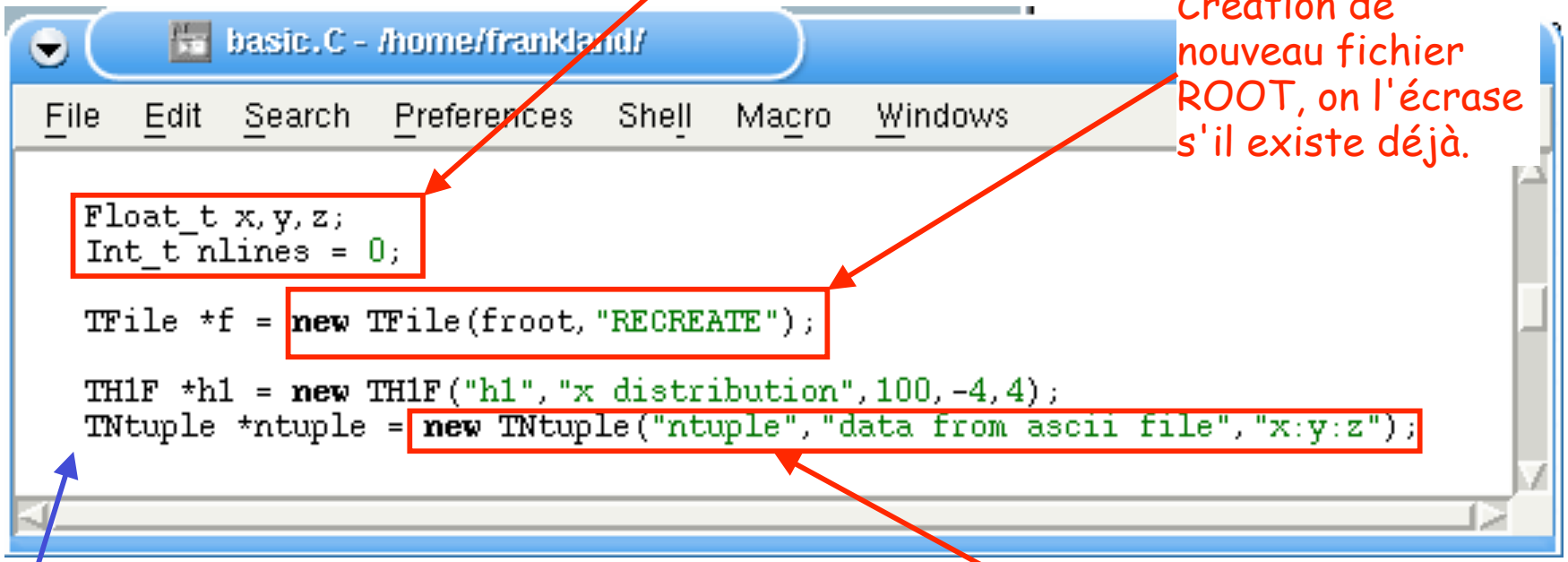
http://caeinfo.in2p3.fr/root/Formation/fr/Jour2/basic.C
http://caeinfo.in2p3.fr/root/Formation/fr/Jour2/basic.dat

Exemple d'analyse

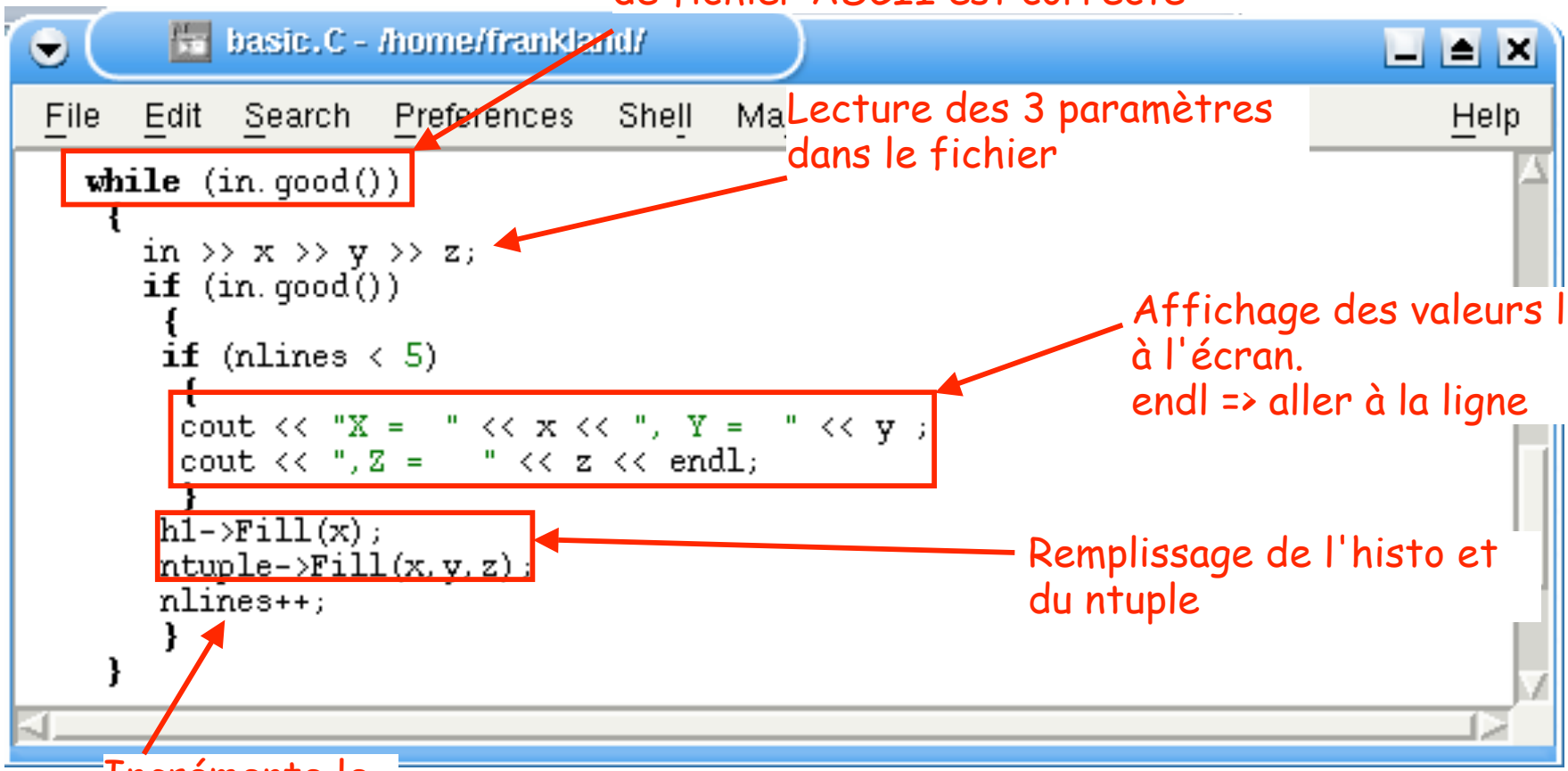
Un exemple de script d'analyse: on lit les données dans un fichier ASCII **basic.dat**; on génère quelques histogrammes que l'on va sauvegarder dans le fichier **basic.root**.



Exemple d'analyse

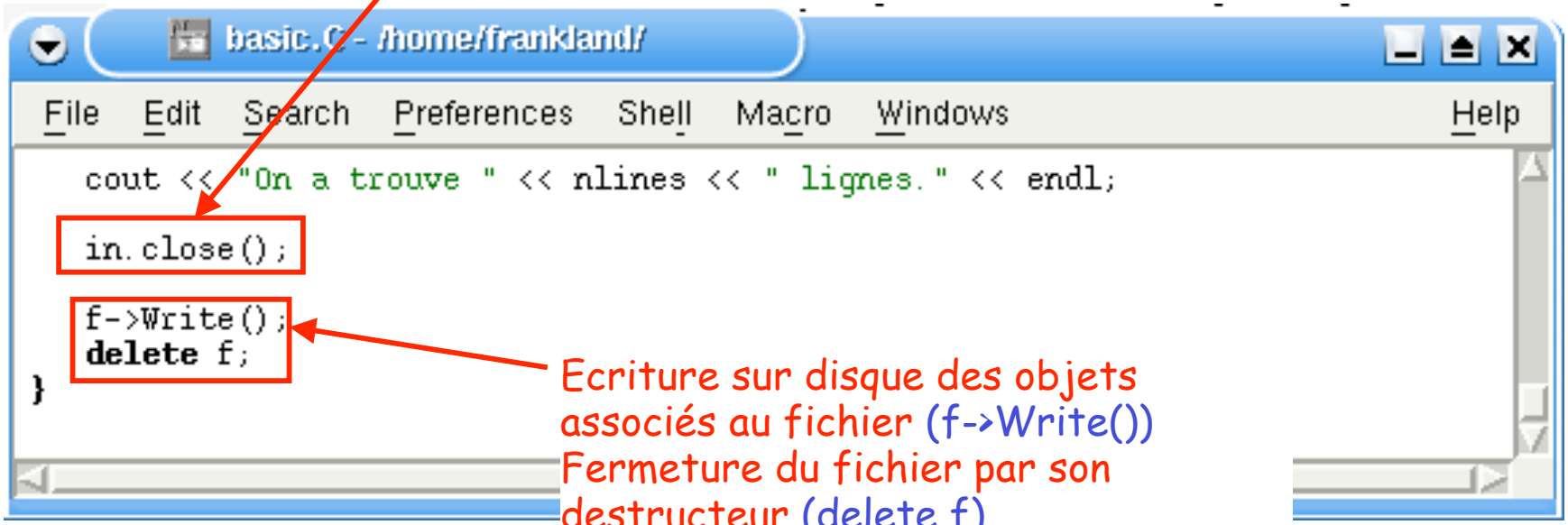


Exemple d'analyse



Exemple d'analyse

Fermeture du fichier ASCII



Ecriture sur disque des objets associés au fichier (f->Write())
Fermeture du fichier par son destructeur (delete f).

WARNING:
les objets associés au fichier n'existent plus en mémoire une fois qu'il a été fermé.

Exécuter l'analyse et voir le résultat

- Compiler et exécuter:

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

- Ouvrir le fichier et afficher le spectre "h1":

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

Affiche le contenu du fichier

Copie en mémoire de l'objet "h1". Retourne l'adresse de cette copie.

Les 2 dernières lignes en 1 seule:

```
fich->Get("h1")->Draw()
```

Dernières astuces

Des exemples de fonctions/scripts

- Sur le site web, rubrique Tutorials, on trouve beaucoup d'exemples
- Attention! S'il n'y a pas d'en-tête avec déclaration d'une fonction, il faut faire:

Exécution d'un script sans en-tête

```
.x toto.C
```

Dans ce cas, le code est interprété au lieu d'être compilé.

WARNING: utilisation fortement déconseillée.

Ce que vous risquez en utilisant l'interpréteur…

L'interpréteur

Vous

Le fichier rootlogon.C

- Ce fichier sans en-tête s'exécute automatiquement lorsqu'on lance ROOT dans le répertoire où il se trouve.

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

Pour avoir de belles couleurs dans les bidims

Retourne la valeur de la variable d'environnement USER

Exécute la commande système "date"

Pour plus d'infos, voir les classes TStyle et TSystem

Exercice

Un autre exemple d'analyse de données

Exercice (Episode 1: The ROOT menace)

http://caeinfo.in2p3.fr/root/Formation/fr/Jour2/exo_j2.data

- Faire un programme d'analyse du fichier **exo_j2.data** dans lequel on a 4 variable x,y,z et e (comme c'est original!)

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

  - Génerer les histogrammes suivants
    - Monodims: distribution de z (**TH1F**)
    - Bidims: y vs x, z vs x, z vs y (**TH2F**)
    - Profiles: <e> vs x, <e> vs y (**TProfile**)
    - Tridims: z vs y vs x, e vs y vs x (**TH3F**)
  - Les sauvegarder dans le fichier **exo_j2.root**.

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

- Ouvrir **`exo_j2.root`**.
- Déterminer les 3 intervalles de *z* les plus peuplés.
  - *Notez-les!*
- En utilisant le FitPanel
  - fitter le TProfile <e> vs x avec un polynôme et *noter* les valeurs des 2 derniers paramètres *ex1* et *ex2*.
  - fitter le TProfile <e> vs y avec un polynôme et *noter* les valeurs des 3 derniers paramètres *ey1*, *ey2* et *ey3*.
- Fermer **`exo_j2.root`**.

Exercice (Part 3: The analysis strikes back)

- Refaire un programme d'analyse du fichier **`exo_j2.data`**.
  - Générer les histogrammes suivants
    - Monodims: distribution (**`TH1F`**) de
      `de=e-ex1*x-ex2*x*x-ey1*y-ey2*y*y-ey3*y*y*y`
    - Bidims: y vs x pour chaque intervalle en z déterminé dans la deuxième partie (**`TH2F`**)
    - Profiles 2D: <z> vs y vs x, <e> vs y vs x (**`TProfile2D`**)
  - Les ajouter au fichier **`exo_j2.root`**.

Exercice
(Part 4: The final shot)

- Déterminer la largeur de la distribution de **`de`** en la fittant avec une gaussienne.
- Ecrire un script permettant de visualiser dans le même TCanvas les quatre histogrammes suivants:
  - y vs x pour z < 1 avec l'option **`col`**
  - y vs x pour 3 < z < 5 avec l'option **`box`**
  - <z> vs y vs x avec l'option **`zcol`**
  - <e> vs y vs x avec l'option **`surf1`**
- Sauvegarder l'image dans un fichier ".gif"