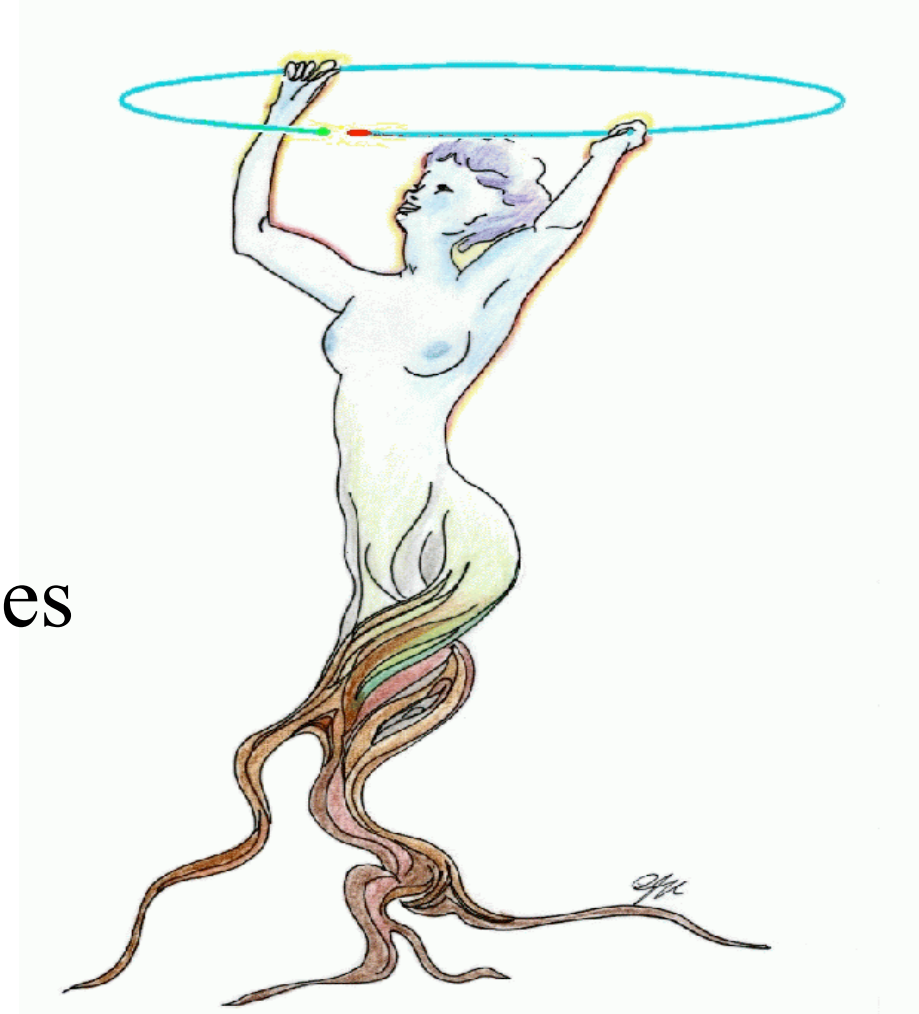
Formation ROOT pour débutants

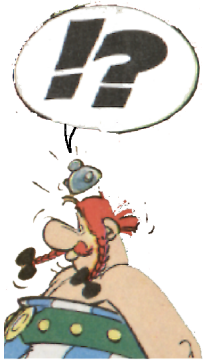
Cinquième Jour

Utilisations avancées



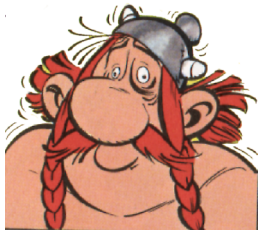
Le meilleur pour la fin

- Utilisation d'un TSelector



- Utilisation d’une classe dans un arbre
- Le polymorphisme

- Ajouter sa classe à ROOT



Les arbres: c'est la classe!

Regrimpons sur l'arbre

- On a vu hier comment ouvrir et manipuler un arbre avec le TreeViewer et les autres ordres (Draw, Scan, SetAlias…)

```
root[0] TFile *f=new TFile("tree_struc.root")
root[1] f->ls()
TFile**         tree_struc.root
 TFile*         tree_struc.root
  KEY: TTree    t;1     TTree avec une structure
root[2] TTree *a=(TTree *)f->Get("t")
root[3] a->StartViewer()
root[4] a->Draw("M_part")
```

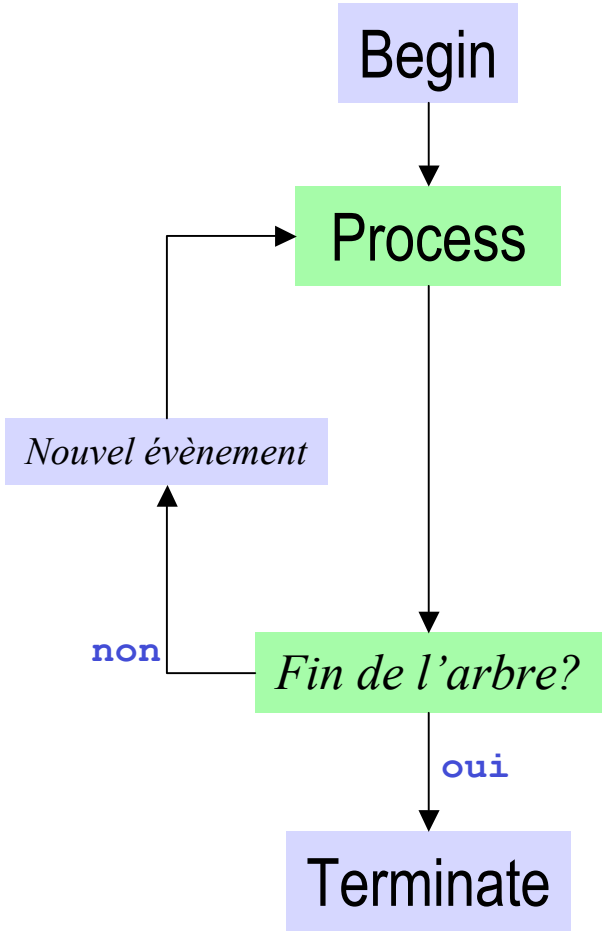
On est bien content?

- Avantage : c'est très simple
- Inconvénient : on ne fait qu'un histogramme à la fois.
- Pour les analyses plus compliquées:
  - Utilisation d'une classe d'analyse

```
root[15] a->MakeSelector("MonSelecteur")
Info in <TTreePlayer::MakeClass>: Files: MonSelecteur.h
and MonSelecteur.C generated from Tree: t
```

Utilisation d'un TSelector

- Il ne faut écrire que 3 méthodes (sous routines)
  - Begin : initialisations diverses (histogrammes, variables globales a l'analyse, etc...)
  - Process : sélection et traitement de l'évènement
  - Terminate : fin de l'analyse (calculs globaux, sauvegarde des résultats, etc…)



Mon premier Begin

(Éditer le fichier `MonSelecteur.C`)

```
#include "MonSelecteur.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void MonSelecteur::Begin(TTree *tree)
{
   // The Begin() function is called at the start of the query.
   // When running with PROOF Begin() is only called on the client.
   // The tree argument is deprecated (on PROOF 0 is passed).

   TString option = GetOption();

   TH1F *h1=new TH1F("hMult","Multiplicite",40,-0.5,39.5);
   TH2F *h2=new TH2F("hEvsZ","Energie vs Z",60,-0.5,59.5,40,0,2400);

}
```

Mon premier Process

```
Bool_t MonSelecteur::Process(Long64_t entry)
{
   // The Process() function is called for each entry in the tree (or possibly
   …
   //  Assuming that fChain is the pointer to the TChain being processed,
   //  use fChain->GetTree()->GetEntry(entry).

   fChain->GetTree()->GetEntry(entry);
   TH1F *h1=(TH1F *)gROOT->FindObject("hMult");
   h1->Fill(M_part);
   TH2F *h2=(TH2F *)gROOT->FindObject("hEvsZ");
   for(Int_t i=0;i<M_part;i++)
    {
    h2->Fill(Z_part[i],E_part[i]);
    }
   return kTRUE;
}
```

Lecture de l'évènement

Les données sont stockées dans des variables qui portent les nom des branches

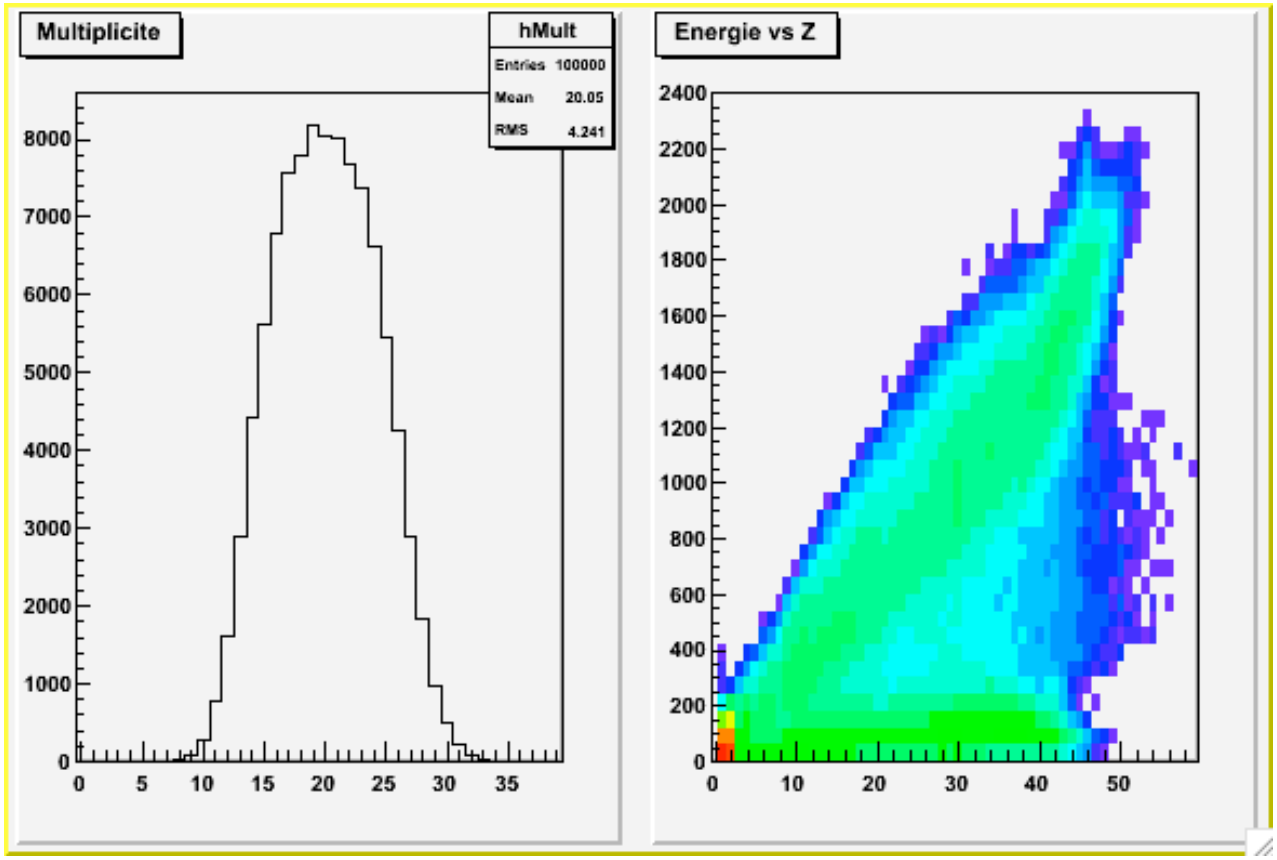
Boucle sur les fragments

Mon premier Terminate

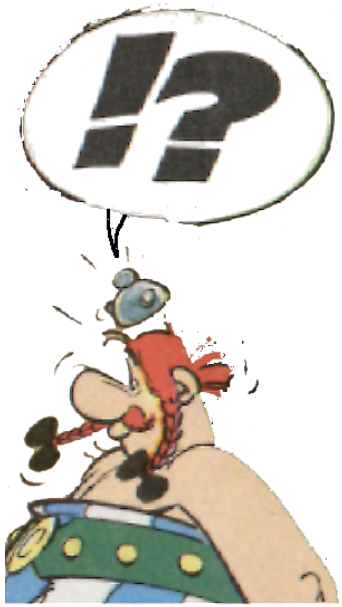
```
void MonSelecteur::Terminate()
{
   // The Terminate() function is the last function to be called during
   // a query. It always runs on the client, it can be used to present
   // the results graphically or save the results to file.

   TCanvas *c=new TCanvas("CanSelecteur","MonSelecteur");
   c->Divide(2,1);          <-- 1 ligne, 2 colonnes
   c->cd(1);
   gROOT->FindObject("hMult")->Draw();
   c->cd(2);
   TH2F *h2=(TH2F *)gROOT->FindObject("hEvsZ");
   h2->SetStats(kFALSE);    <-- Pas de statistiques pour le bidim
   h2->Draw("col");
   gPad->SetLogz(kTRUE);    <-- Echelle logarithmique pour l'axe Z
   c->Update();

}
```

Exécution!

```
root[19] a->Process("MonSelecteur.C+")
Info in <TUnixSystem::ACLiC>: creating shared library ./MonSelecteur_C.so
Class MonSelecteur: Streamer() not declared
Class MonSelecteur: ShowMembers() not declared
```



Utilisation d'une classe dans un arbre



Un arbre plus compliqué: utilisation de classes

- On va utiliser 2 classes
  - Une classe **`Fragment`** qui va contenir les informations relatives à 1 particule (fichiers **`Fragment.h`** et **`Fragment.C`**)
  - Une classe **`Event`** qui va contenir un tableau de particules et des info générales sur l'évènement (fichiers **`Event.h`** et **`Event.C`**)

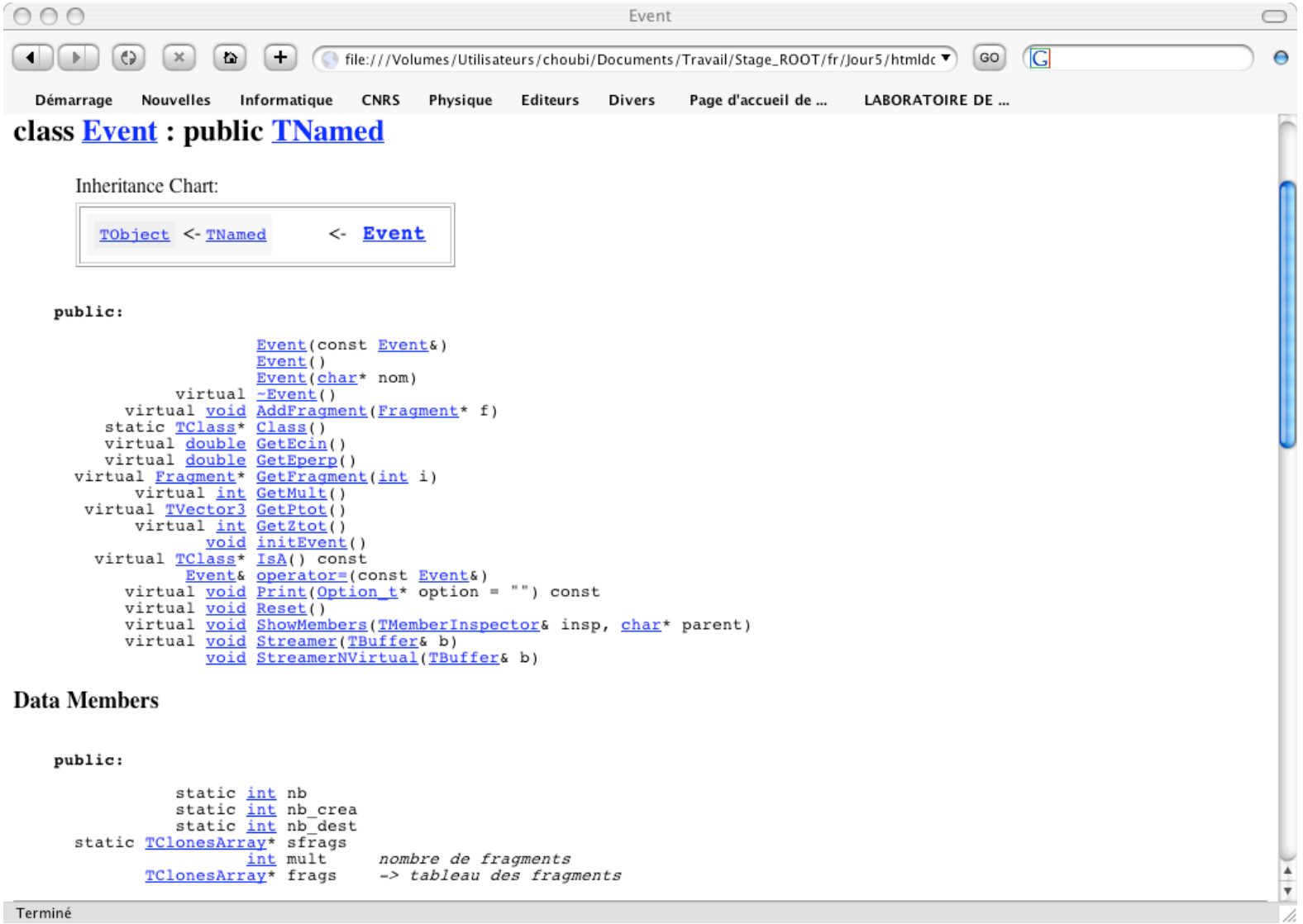
`http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/Fragment.*`
`http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/Event.*`

La classe Fragment

http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/htmldoc/Fragment.html

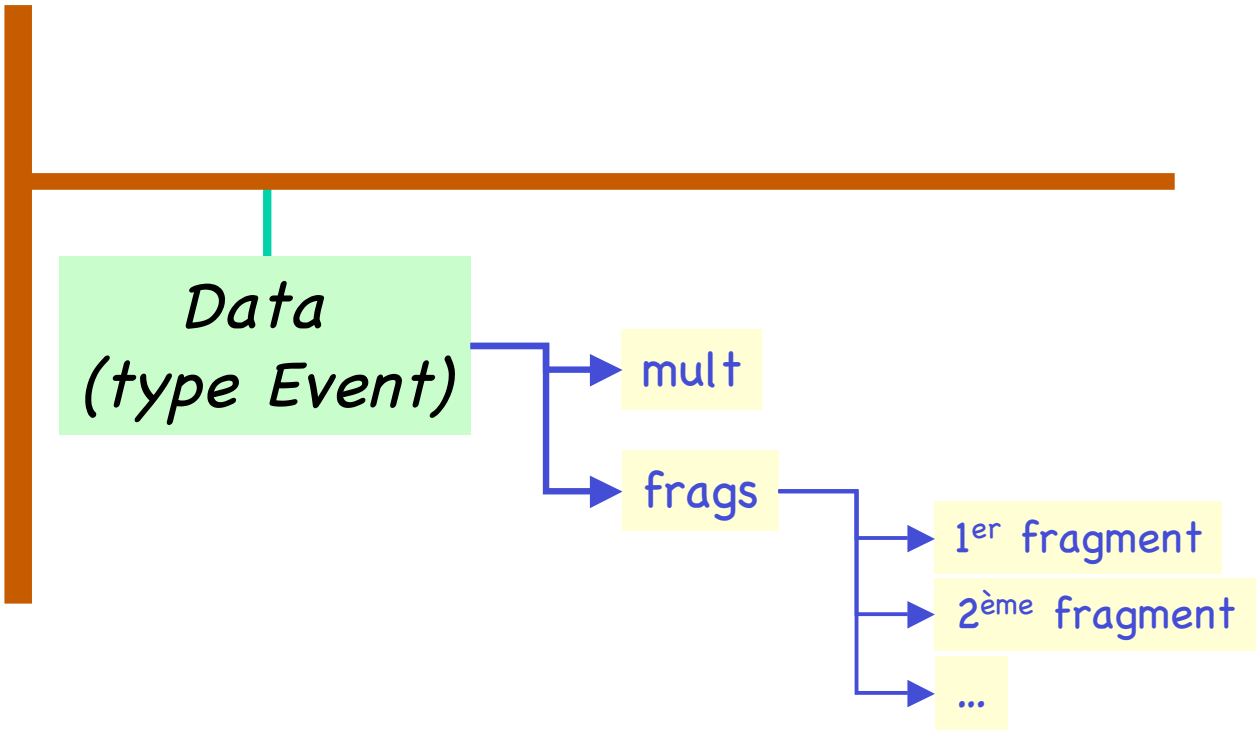
La classe Event

http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/htmldoc/Event.html



Structure de l'arbre

- L'arbre contient 1 branche avec 1 feuille de type Event:



(Pour voir comment il a été fabriqué, regarder FillArbre.C)
http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/FillArbre.C

Pour l'utiliser

- Il faut charger les objets dans ROOT

```
root[1] .L $ROOTSYS/lib/libPhysics.so
root[2] .L Fragment.C+
root[3] .L Event.C+
```

Nécessaire pour les TVector3 utilisés dans la classe Fragment

- Il faut ouvrir le fichier

```
root[4] TFile *fi=new TFile("indra_xesn50.root")
```

- Il faut initialiser l'arbre

```
root[5] TTree *mt=(TTree *)fi->Get("Arbre")
root[6] Event *evt=new Event()
root[7] mt->SetBranchAddress("Data",&evt)
```

Les données lues seront stockées dans l'événement pointé par ***evt***

- C'est prêt!

http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/indra_xesn50.root

Fichier rootlogon.C

Uniquement pour les maladroits ou les fainéants!

```
{
gStyle->SetPalette(1);
gROOT->ProcessLine(".L $ROOTSYS/lib/libPhysics.so");
gROOT->ProcessLine(".L Fragment.C+");
gROOT->ProcessLine(".L Event.C+");
TFile *fi=new TFile("indra_xesn50.root");
TTree *mt=(TTree *)fi->Get("Arbre");
Event *evt=new Event();
mt→SetBranchAddress("Data",&evt);
}
```

Lire un évènement dans l'arbre

```
root[8] mt->GetEntries()
```
Nombre total d'évènements dans l'arbre

```
1.3201100000000000e+05
root[9] mt->GetEntry(1567)
```
Lecture de l'évènement 1567 dans l'arbre

```
(Int_t)1089
root[10] evt->Print()
```
Listing de l'évènement lu

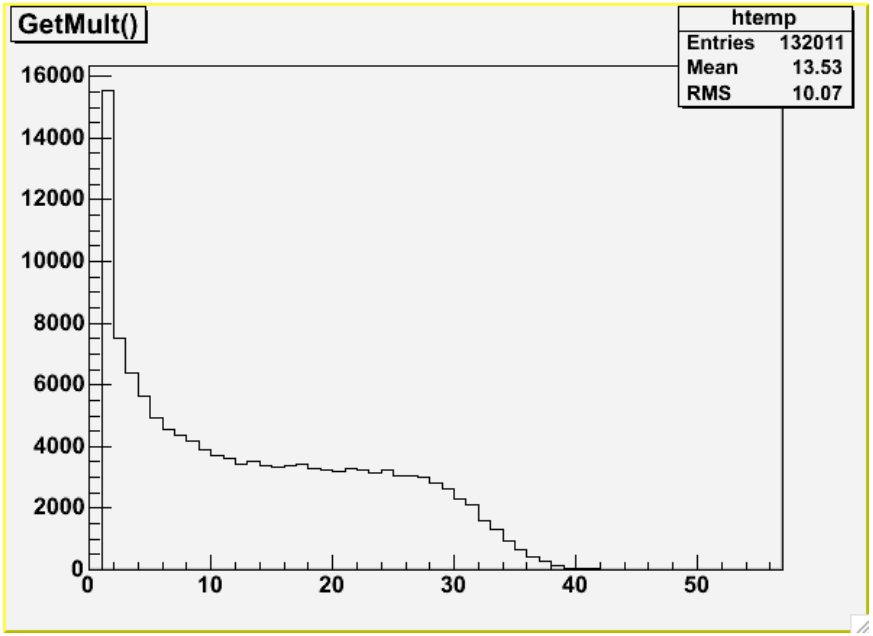
```
==========================================================
Mult : 21
1 ->   8,   4 :     -0.47     -0.22      2.27
2 ->   4,   2 :      0.97      2.43      8.58
3 ->   1,   1 :      0.22      4.84     12.56
4 ->   4,   2 :     -0.25     -2.33      8.96
…
```

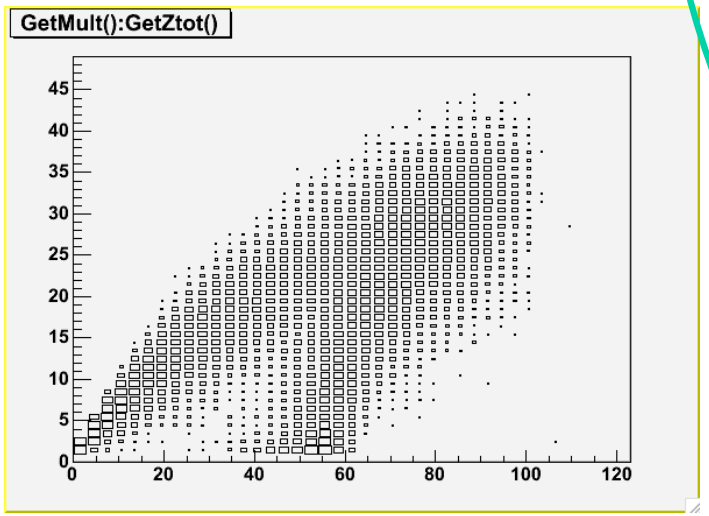
```
root[11] evt->GetEperp()
```
Énergie perpendiculaire de l'évènement

```
(double)2.899……e+02
```

Faire un histogramme de variables (Et 1)

```
root[12] mt->Draw("GetMult()")
```



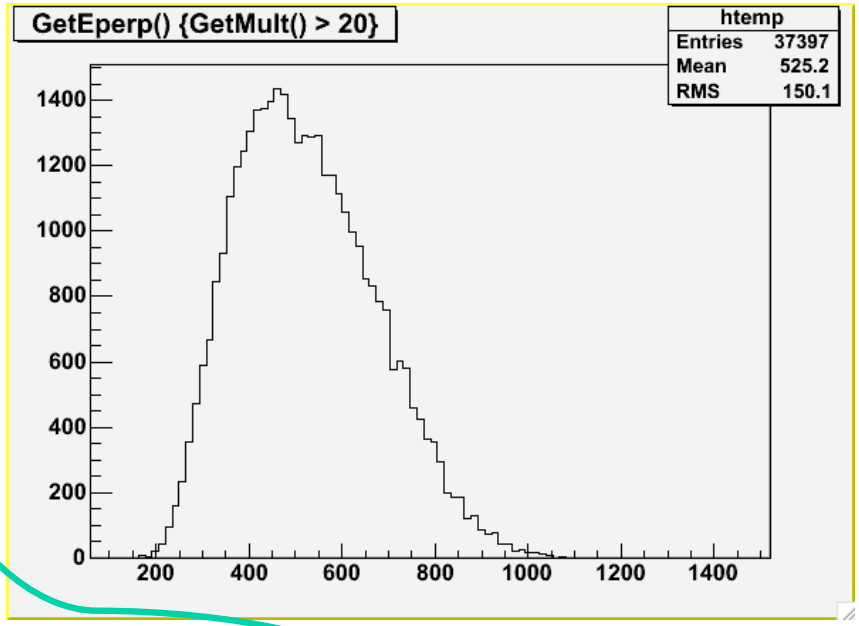


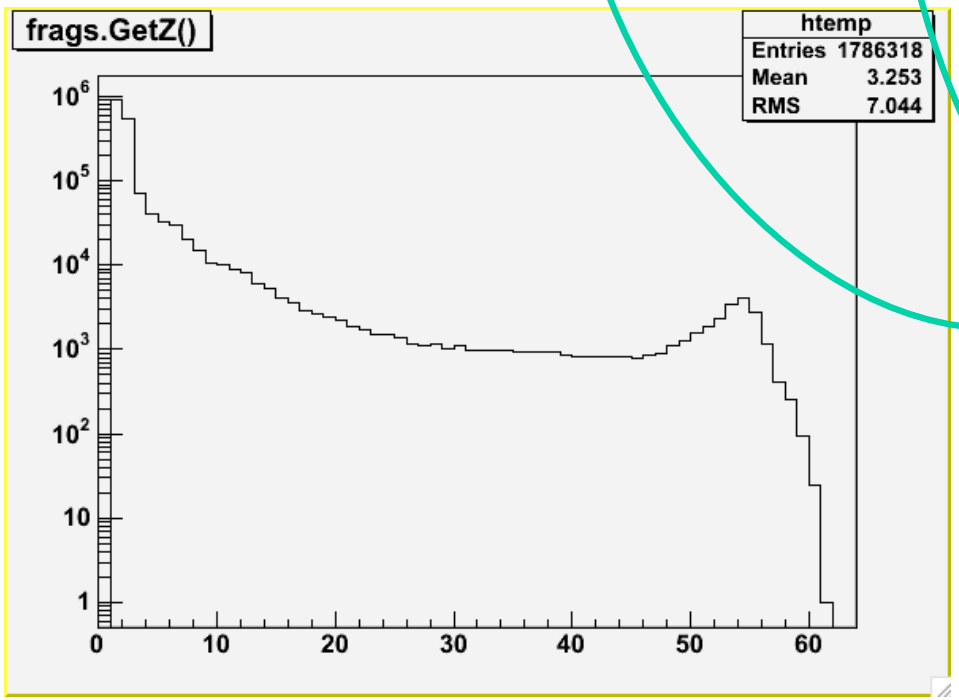

```
root[13] mt->Draw("GetMult():GetZtot()","","box")
```

Ce sont des chaines de caractères!
Pas du C++!

Faire un histogramme de variables (Et 2)

```
root[14] mt->Draw("GetEperp()","GetMult()>20")
```





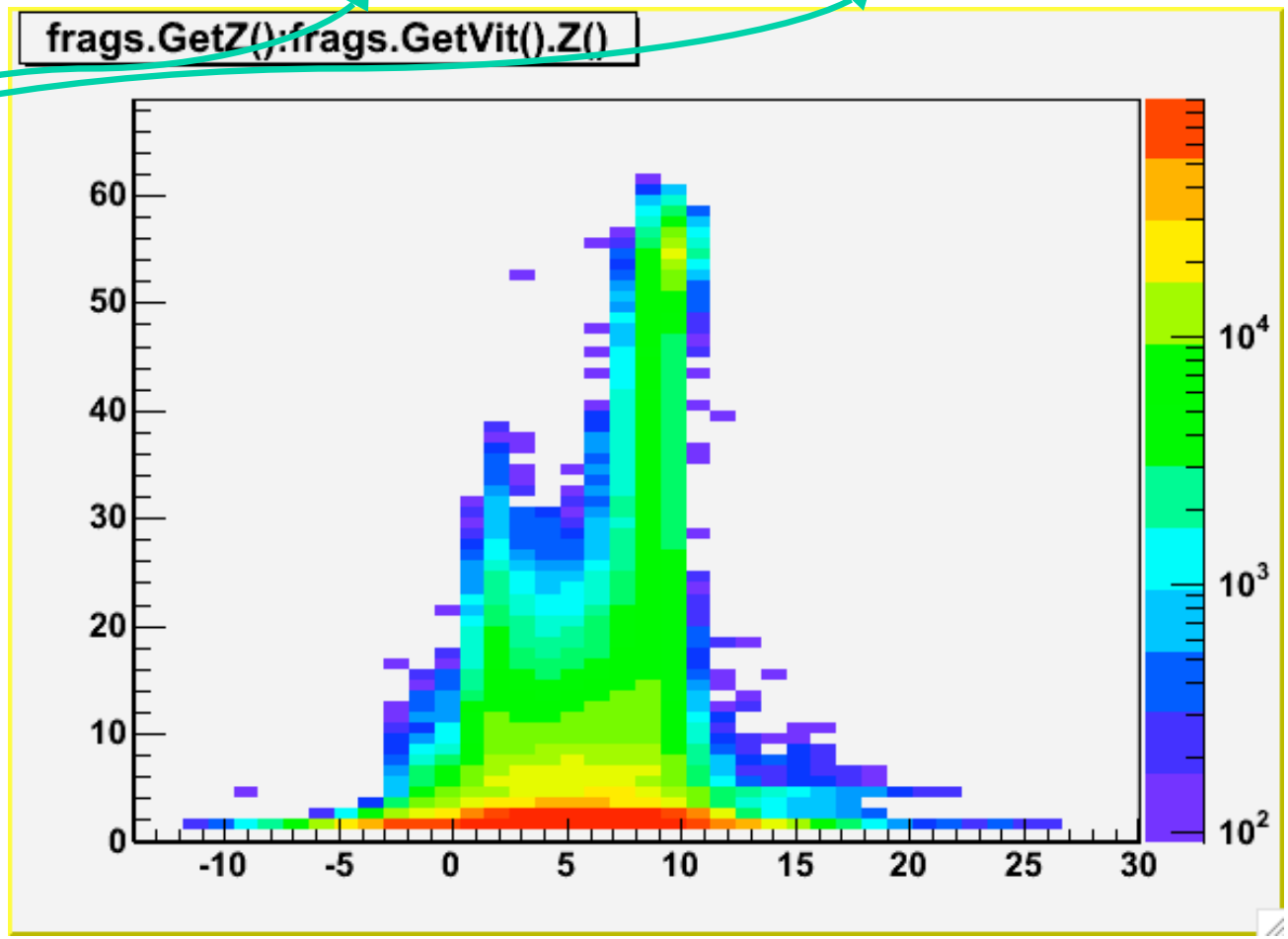
Ce sont des chaines de caractères!
Pas du C++!

```
root[15] mt->Draw("frags.GetZ()")
```

Faire un histogramme de variables (Et 3)

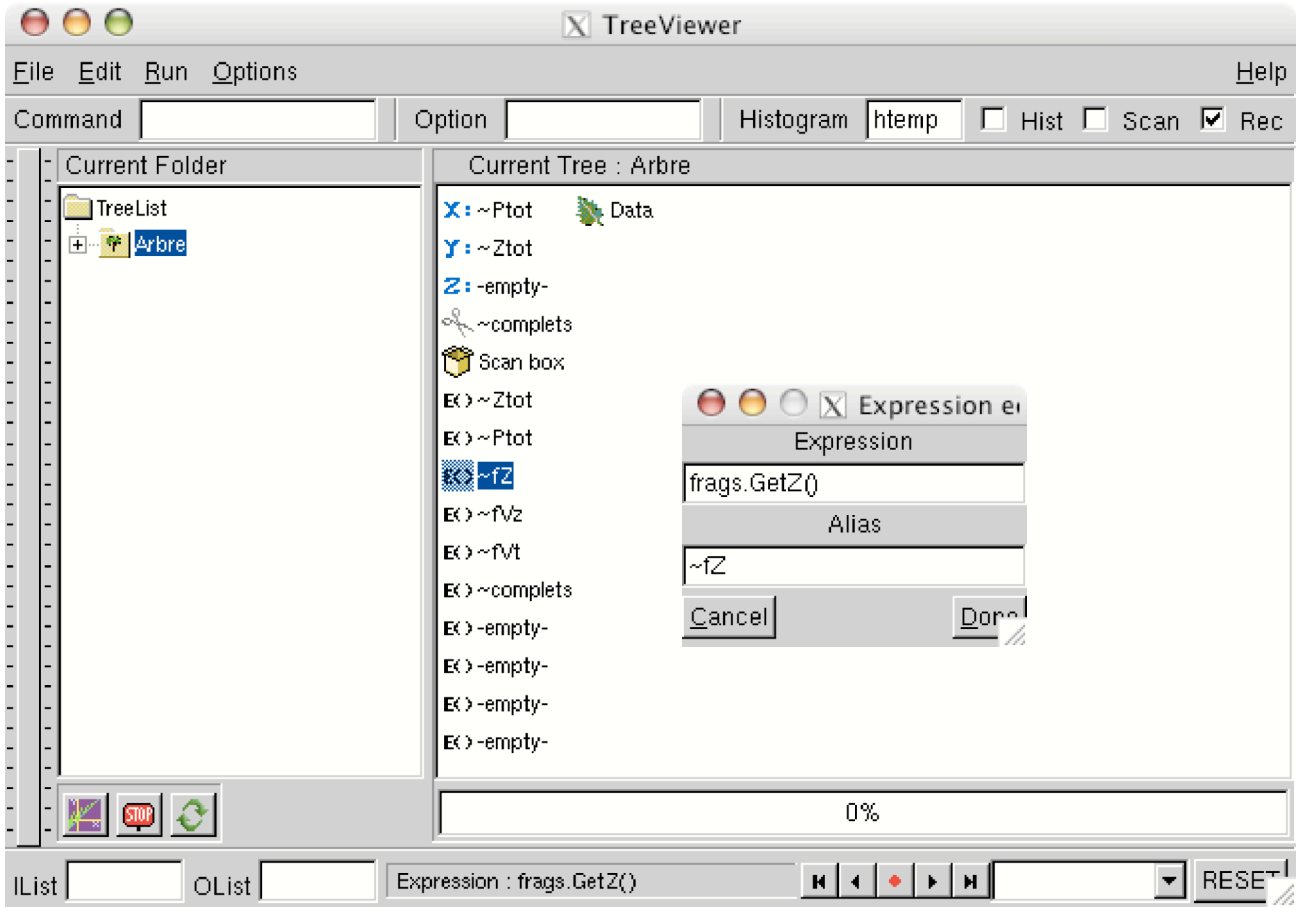
```
root[16] mt->Draw("frags.GetZ():frags.GetVit().Z()","","zcol")
```

Ce sont des chaines de caractères! Pas du C++!



Utilisation du TTreeViewer

```
root[17] mt->StartViewer()
```

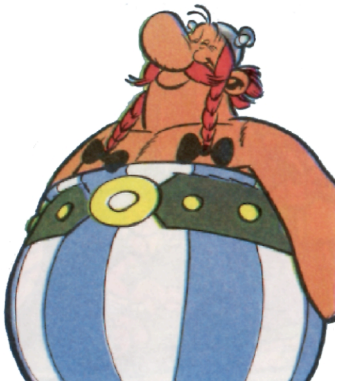


On peut utiliser des coupures TCut et TCutG

Utilisation dans un TSelector

- Comme c'est un arbre, on peut aussi utiliser un TSelector:

```
root[18] mt->MakeSelector("MonAnalyse")
Info in <TTreePlayer::MakeClass>: Files: MonAnalyse.h
and MonAnanlyse.C generated from Tree: Arbre
```



Mon second Begin

```
#include "Fragment.h"
#include "Event.h"
#include "MonAnalyse.h"
#include "TH2.h"
#include "TStyle.h"
#include "TCanvas.h"

void MonAnalyse::Begin(TTree *tree)
{
   // The Begin() function is called at the start of the query

   TString option = GetOption();
    // Declaration des histogrammes
    TH2F *h2=new TH2F("ZtPt","Ztot vs Ptot",40,0,800,60,0,120);
    TH1F *h1=new TH1F("distZ","Z",120,0,120);
}
```

Ajout indispensable !

(Éditer le fichier `MonAnalyse.C`)

http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/MonAnalyse.C

Mon second Process

```
Bool_t MonAnalyse::Process(Long64_t entry)
{
   // Function called for selected entries only.
   // Entry is the entry number in the current tree.
   // Read branches not processed in ProcessCut() and fill histograms.
   // To read complete event, call fChain->GetTree()->GetEntry(entry).

   fChain->GetTree()->GetEntry(entry);
   TH2F *h2=(TH2F *)gROOT->FindObject("ZtPt");
   h2->Fill(Data->GetPtot().Z(),Data->GetZtot(),1.);
   TH1F *h1=(TH1F *)gROOT->FindObject("distZ");
   for(Int_t i=1;i<=Data->GetMult();i++)
        {
        Fragment *fra=Data->GetFragment(i);
        h1->Fill(fra->GetZ(),1.);
        }
   return kTRUE;
}
```

Lecture de l'évènement

L'évènement lu est stocké dans la variable Data (Event) car c'est le nom de la branche.

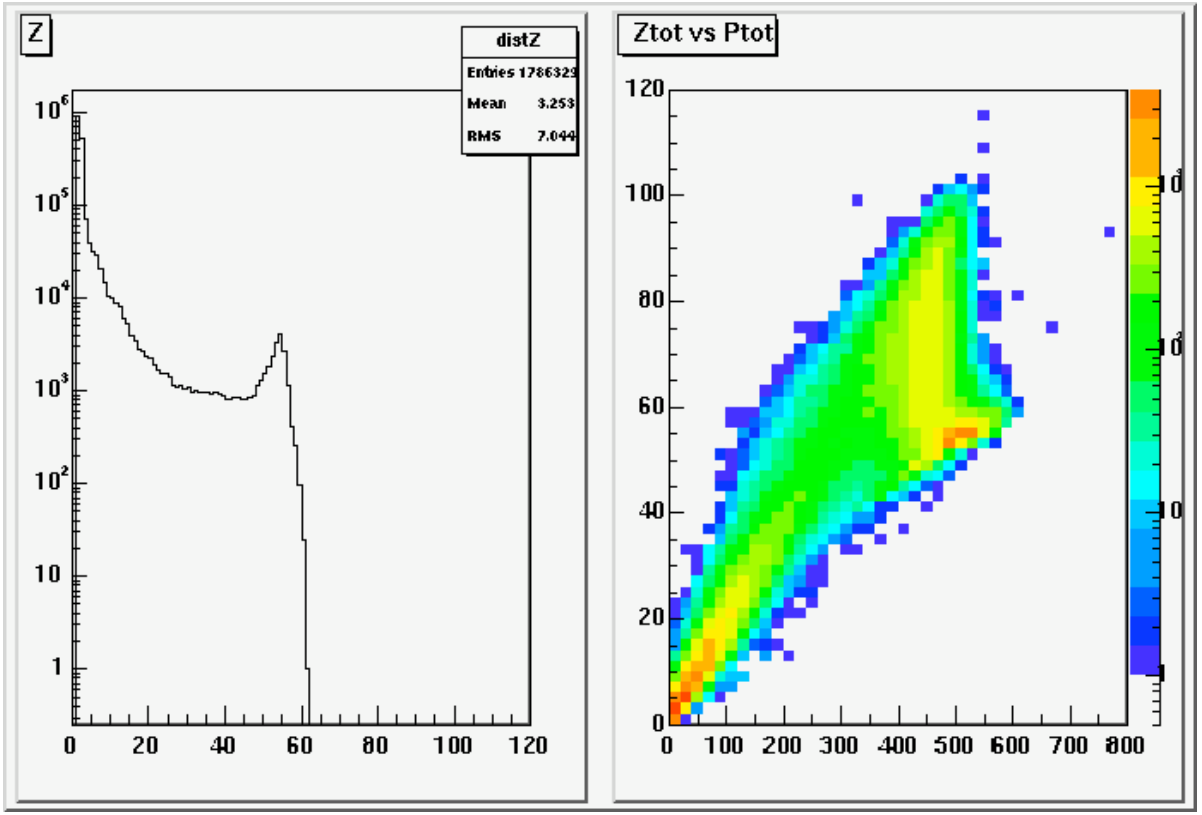
Boucle sur les fragments

Mon second Terminate

```
void MonAnalyse::Terminate()
{
   // Function called at the end of the event loop.
   // On affiche les spectres
   TH2F *h2=(TH2F *)gROOT->FindObject("ZtPt");
   TH1F *h1=(TH1F *)gROOT->FindObject("distZ");
   TCanvas *c2=(TCanvas *)gROOT->FindObject("c2");
   if(!c2)
    {
    c2=new TCanvas("c2","Resultat");
    }
   c2->Clear();
   c2->Divide(2,1);
   c2->cd(1);h1->SetStats(kTRUE);h1->Draw(); gPad->SetLogy(kTRUE);
   c2->cd(2);h2->SetStats(kFALSE);h2->Draw("zcol");gPad->SetLogz(kTRUE);
   c2->Update();
}
```

Exécution!

```
root[19] mt->Process("MonAnalyse.C+")
Info in <TUnixSystem::ACLiC>: creating shared library ./MonAnalyse_C.so
Class MonAnalyse: Streamer() not declared
Class MonAnalyse: ShowMembers() not declared
```



On a perdu les sources de Event et de Fragment!

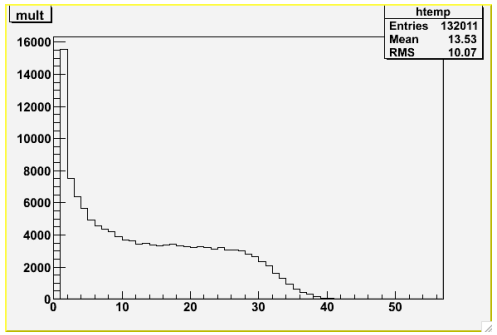
On va regénérer l'essentiel: l'accès aux champs (variables) de base.

```
root[0] TFile *f=new TFile("indra_xesn50.root")
Warning in <TClass::TClass>: no dictionary for class Event is available
Warning in <TClass::TClass>: no dictionary for class Fragment is available
Warning in <TClass::TClass>: no dictionary for class TVector3 is available
root[1] f->MakeProject("indra","*","recreate++")
MakeProject has generated 3 classes in indra
indra/MAKE file has been generated
Shared lib indra/indra.so has been generated
Shared lib indra/indra.so has been dynamically linked
root[2] .class Event
List of member variable----------------------------------------------------
Defined in Event
indra/indra.so    1 0x1b       int mult //nombre de fragments
indra/indra.so    1 0x1f       TClonesArray* frags //-> tableau des fragments
root[3] .class Fragment
List of member variable----------------------------------------------------
Defined in Fragment
indra/indra.so    1 0xb        int A //nombre de nucleons
indra/indra.so    1 0xf        int Z //nombre de charges
indra/indra.so    1 0x13       TVector3 v , size=40 //vitesse
indra/indra.so    1   0xf        Double_t fX
indra/indra.so    1   0x17       Double_t fY
indra/indra.so    1   0x1f       Double_t fZ
```

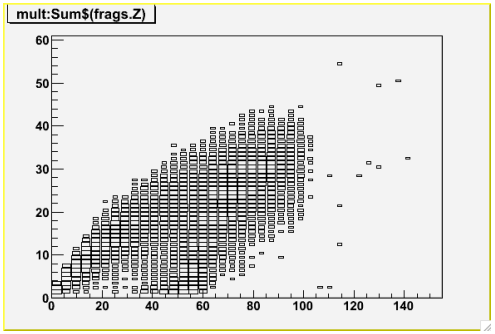
A la recherche des classes perdues (suite…)

```
root[4] TTree *mt=(TTree *)f->Get("Arbre")
```

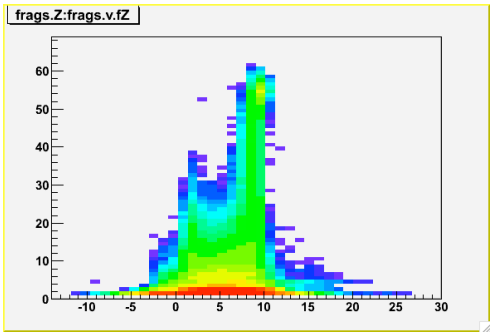
```
root[5] mt->Draw("mult")
```




```
root[6] mt->Draw("mult:Sum$(frags.Z)","","box")
```




```
root[7] mt->Draw("frags.Z:frags.v.fZ","","col")
```



Le polymorphisme

Prendre des vessies pour des lanternes (sans se bruler!)

- Voici un exemple tout simple:

```
root[20] TList *tl=new TList()
root[21] TF1 *f1=new TF1("fun","gaus",0,100)
root[22] f1->SetParameters(70,15,2)
root[23] TF1 *f2=new TF1("fdeux","gaus",0,200)
root[24] f2->SetParameters(50,65,8)
root[25] TArc *arc=new TArc(10,20,10)
root[26] tl->Add(f1)
root[27] tl->Add(f2)
root[28] tl->Add(arc)
root[29] tl->ls()
```

Liste de TObject

Est-ce correct ?

- Que se passe-t-il ?

Comment ça marche?

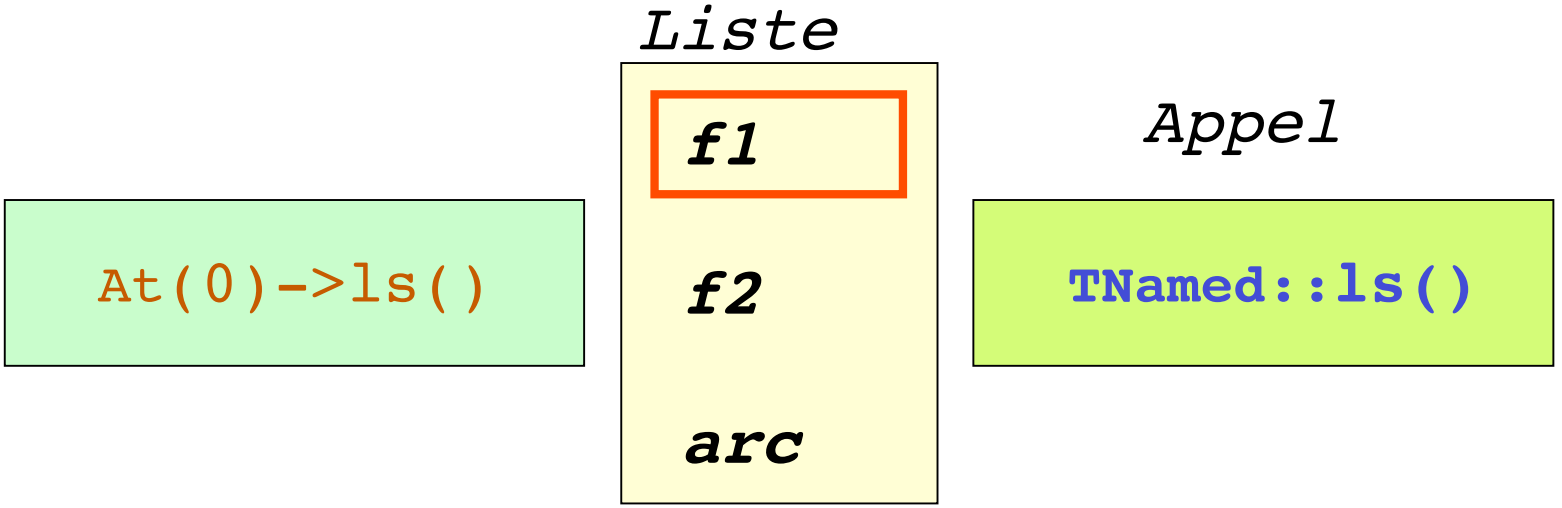
- En fait, les objets ajoutés sont tous des **`TObject`** ayant chacun sa propre méthode **`ls`**
- **`tl->ls()`** exécute la méthode **`ls`** pour chaque élément. La "bonne" méthode est sélectionnée automatiquement

```
TList::ls(void)
{
 for(int i=0;i<GetSize();i++)
 {
 At(i)->ls();
 }
}
```

nombre d'éléments dans la liste

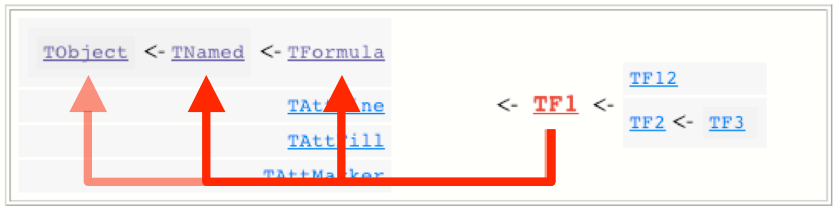
Retrouve le $i^{ème}$ élément (TObject) dans la liste

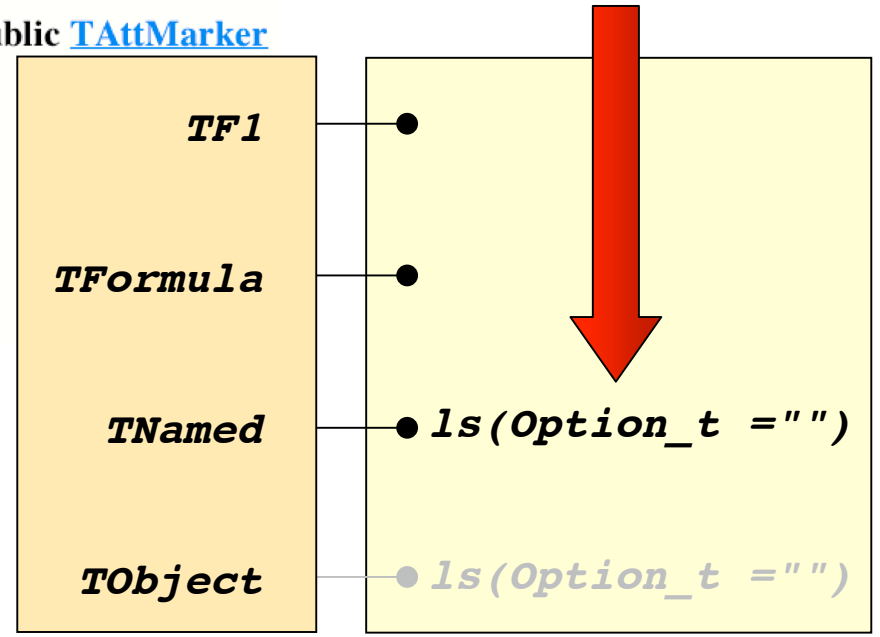
Ce qui se passe en mémoire



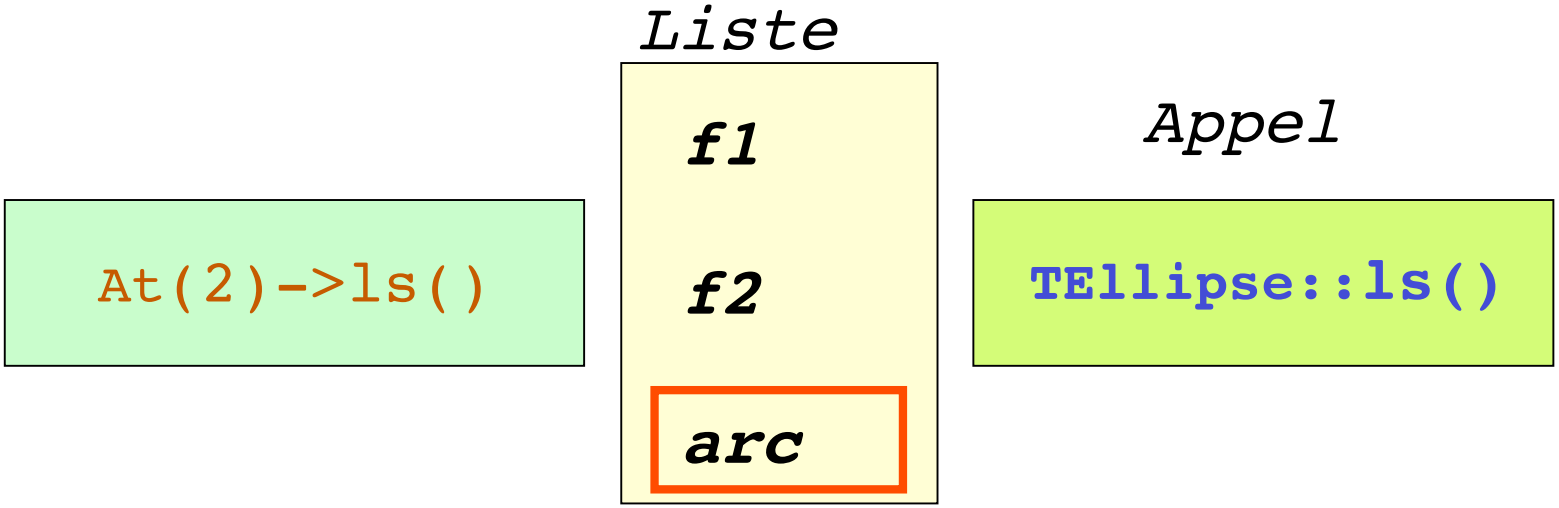
class TF1 : public TFormula, public TAttLine, public TAttFill, public TAttMarker

Inheritance Chart:



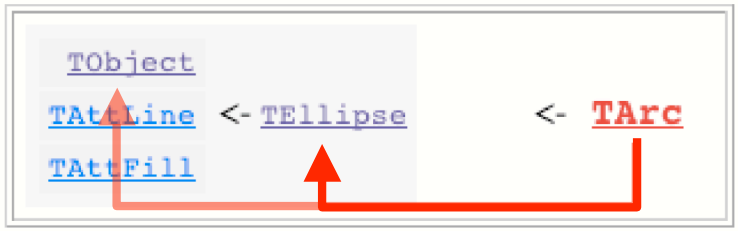


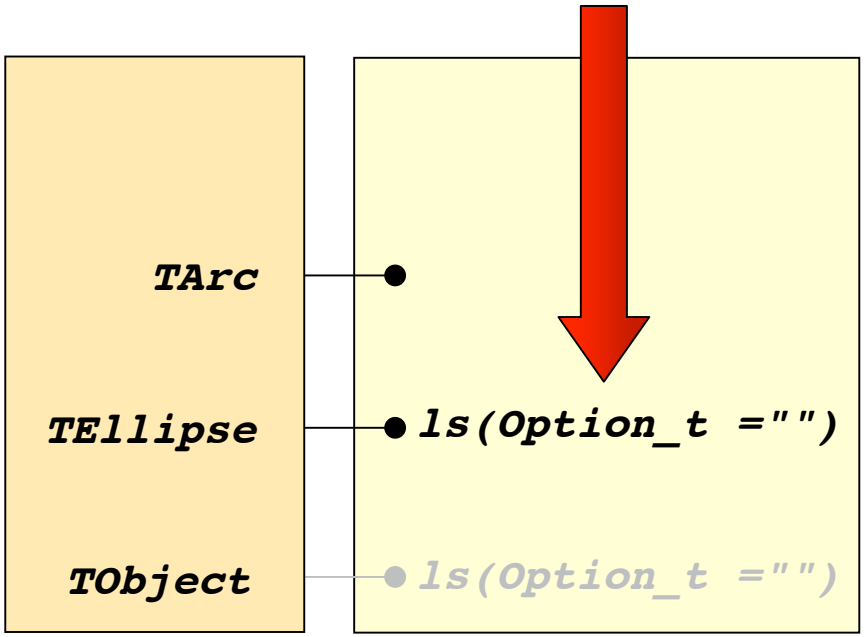
Ce qui se passe en mémoire



class TArc : public TEllipse

Inheritance Chart:





On aurait pu le faire en fortran!

```
Subroutine ListeLs(nb_element,elementId,element)
    do i=1,nb_element
     if(elementId(i).eq.idTF1) then
       call TF1Ls(element(i))
     else if(elementId(i).eq.idArc) then
       call ArcLs(element(i))
     else if(elementId(i).eq.idLatex) then
       call LatexLs(element(i))
     endif
    enddo
    return
    end
```

- Cela devient vite compliqué si on veut ajouter d'autres objets, alors que `TList::ls()` est écrite une fois pour toutes!

Un autre exemple: dessins des objets d'une TList

```
root[30] tl->Draw()
```

- Que se passe-t-il ?

```
TList::Draw(Option_t *opt)
{
 for(int i=0;i<GetSize();i++)
 {
 At(i)->Draw(opt);
 }
}
```

- Détail de la boucle

```
At(0)->Draw() ≡ f1->Draw()
At(1)->Draw() ≡ f2->Draw()   <-- Efface le premier!
At(2)->Draw() ≡ arc->Draw()
```

Modifier l'affichage d'une TList tout en gardant le reste...

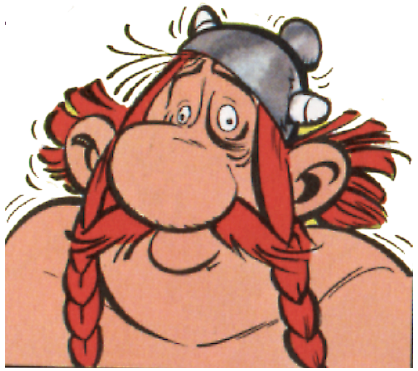
- Ce que l'on voudrait, c'est avoir une `TList` pour laquelle `Draw()` affiche le premier élément avec l'option demandée, et que les autres éléments s'affichent avec l'option `"same"` .
- C'est possible grâce à l'héritage!

```
Class MaListe: public TList
```

- On ne va redéfinir que la méthode `Draw()`

```
void MaListe::Draw(Option_t *opt="")
```

Intégrer une nouvelle classe MaListe à ROOT

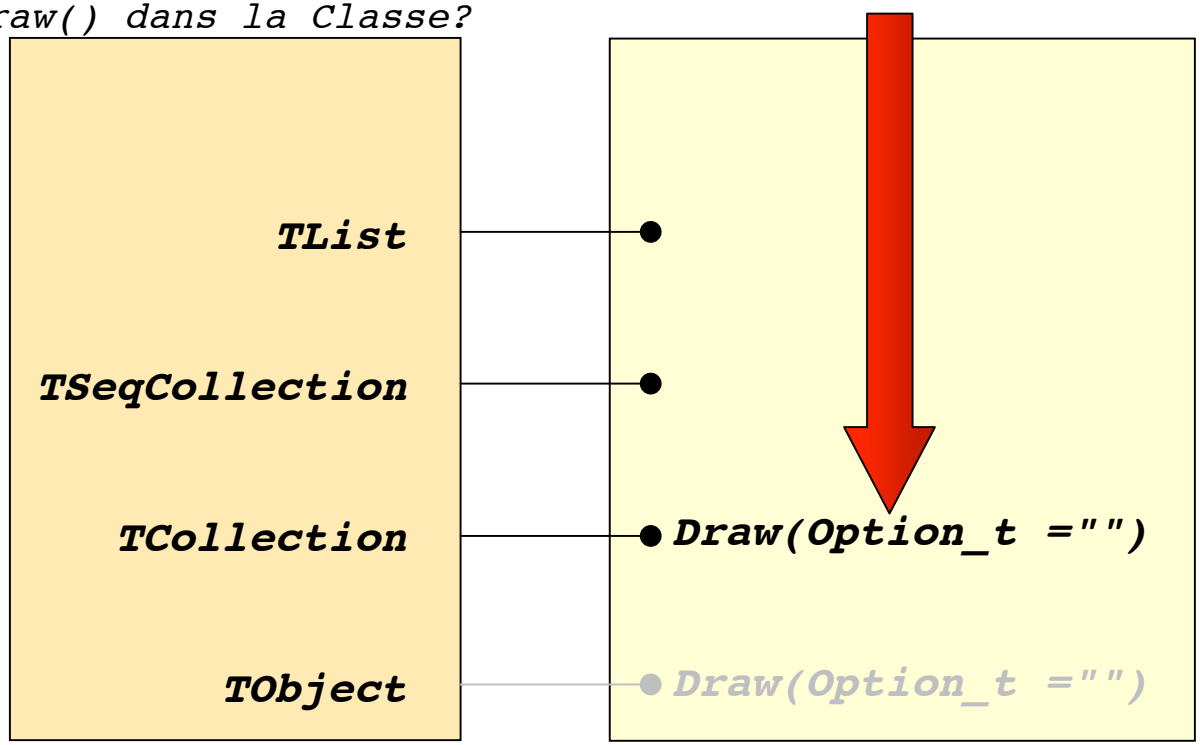


Quelle méthode Draw()?

class TList : public TSeqCollection

Inheritance Chart:

Draw() dans la Classe?

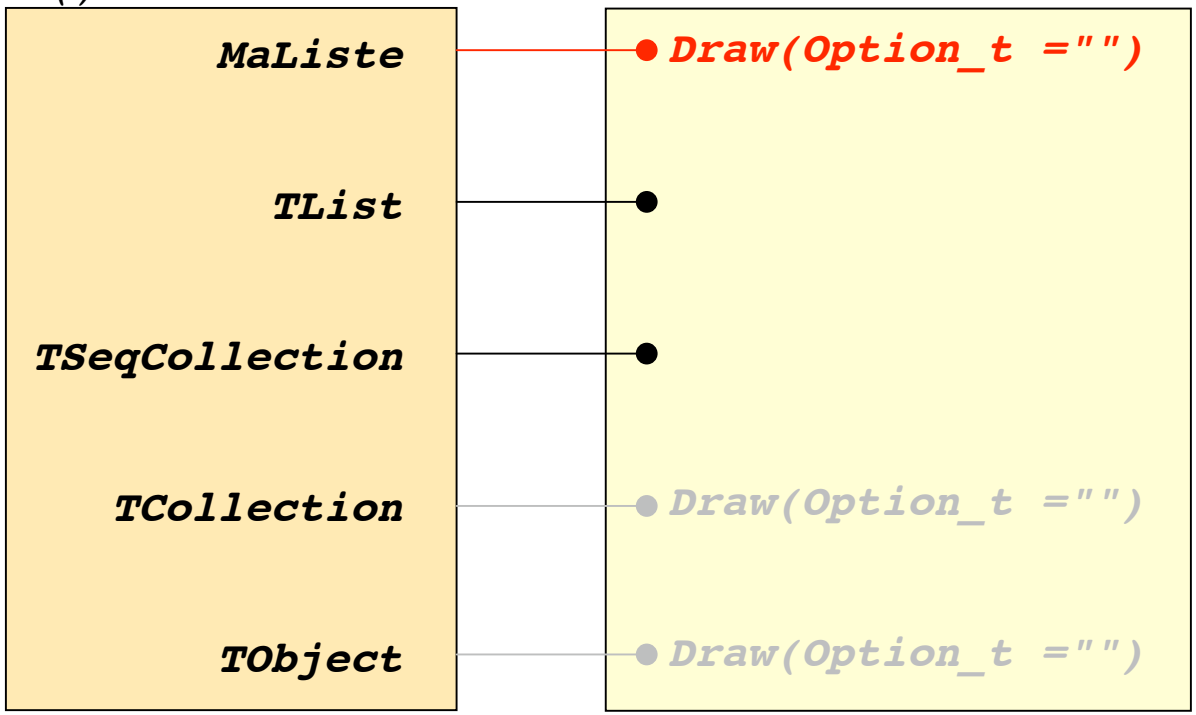


Quelle méthode Draw()?

class TList : public TSeqCollection

Inheritance Chart:

Draw() dans la Classe?



Ma classe à moi MaListe
I: Définition

Fichier **MaListe.h**

```
//
// Définition de MaListe
//
//

#ifndef MaListe_h
#define MaListe_h
#include "TList.h"

class MaListe:public TList
        {
// Champs Statiques
// Champs
// Methodes
        public:
        MaListe(void);                  // constructeur par defaut
        virtual ~MaListe(void);         // destructeur

        virtual void Draw(Option_t *opt=""); // Methode de dessin

        ClassDef(MaListe,1)

        };
#endif
```

Obligatoire!

Définition IDENTIQUE à celle de TCollection

Recommandé pour ROOT!

http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/MaListe.h

Ma classe à moi MaListe
II: Implémentation

```
#include "MaListe.h"


ClassImp(MaListe)

MaListe::MaListe(void):TList()
{
//
// constructeur par defaut
//
}
MaListe::~MaListe(void)
{
//
// Destructeur
//
}
```

Recommandé pour ROOT!

```
void MaListe::Draw(Option_t *opt)
{
//
// Méthode de dessin
//
 for(int i=0;i<GetSize();i++)
   {
    if(i == 0)
    {
    At(i)->Draw(opt);
    }
   else
    {
    At(i)->Draw("same");
    }
   }
 }
```

Fichier **MaListe.C**

http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/MaListe.C

Ma classe à moi MaListe
III: Utilisation

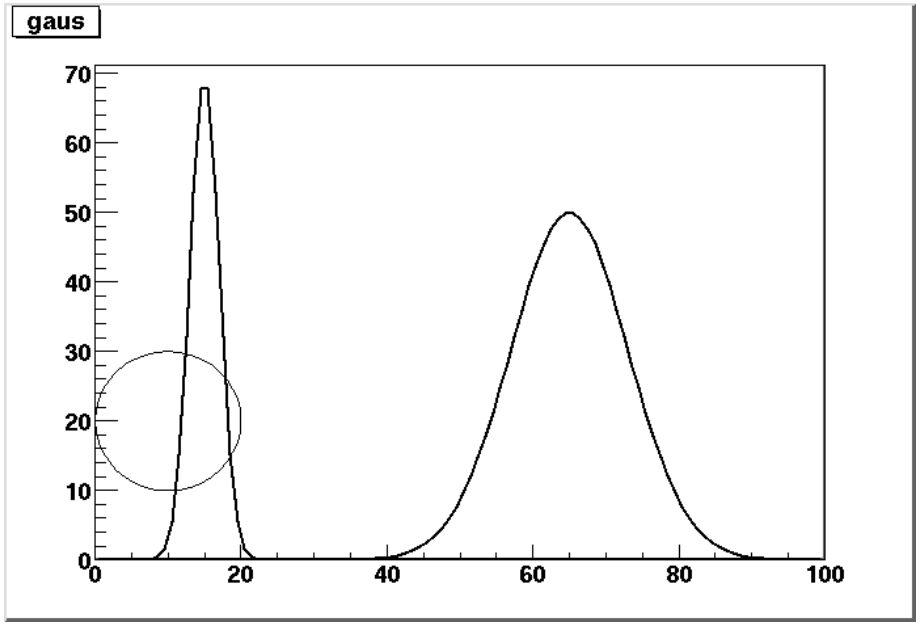
- Pour l'intégrer dans ROOT:

```
root[30] .L MaListe.C+
```

- Pour l'utiliser

```
root[31] MaListe *ml=new MaListe()
root[32] ml->AddAll(tl)
root[33] ml->Draw()
```

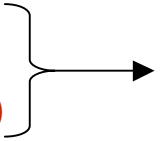
- C'est gagné !



Ma classe à moi MaListe
IV: Amusons nous un peu

- Essayer:

```
root[40] gPad->DrawFrame(0,0,80,200)
root[41] ml->Draw("same")
root[42] .class MaListe
root[43] ml->DrawClass()
root[44] ml->ls()
root[45] ml->Inspect()
root[46] ml->ClassName()
root[47] THtml *htm=new THtml()
root[48] htm->MakeClass("MaListe")
```



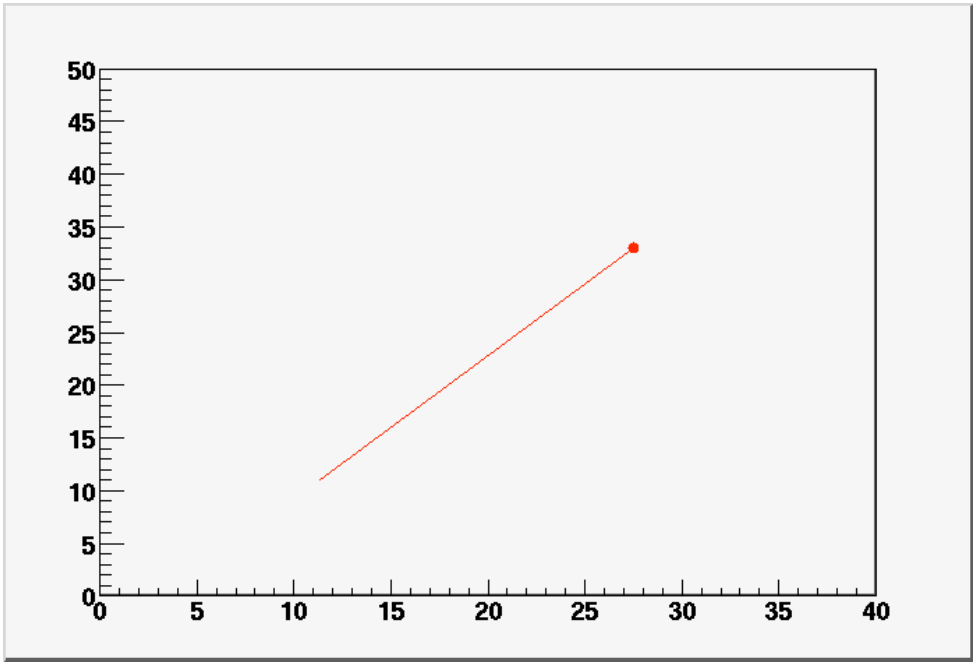
Génération de la documentation!

Exercice 1

- En utilisant un TSelector, fabriquer les histogrammes suivants pour les évènements de **`indra_xesn50.root`** dont la charge totale (Z_{tot}) est supérieure à 80:
  - Z vs V_z (regarder les méthodes de **`TVector3`**)
  - Z_{max} (Z le plus grand de l'évènement) vs E_{perp}
  - $V_{transverse}$ vs V_z pour Z=2 et Z=6 (regarder **`TVector3`**)
  - <Z>
- Sauvegarder le résultat dans le fichier **`resultat.root`**.

Exercice 2

A partir de la classe **`TLine`**, fabriquer une classe **`MPointeur`** qui affiche en bout de ligne un cercle plein, comme dans l'exemple ci-dessous en surchargeant la méthode **`Paint()`** . On utilisera pour le dessin du cercle la classe **`TMarker`** avec le style 20. La ligne sera dessinée avec l'instruction **`TLine::Paint()`**. Penser à faire deux constructeurs: **`MPointeur()`** et **`MPointeur(x1,y1,x2,y2)`** (regarder ceux de **`TLine`**).



Application ROOT

```
#include "TH1F.h"
#include "TApplication.h"
#include "TRint.h"

int main(int argc, char *argv[])
{
#ifdef WITHRINT
TRint *myapp=new TRint("RootSession",&argc,argv,NULL,0);
#else
TApplication *myapp=new TApplication("myapp",0,0);
#endif
TH1F *h=new TH1F("h","Test",100,-10,10);
h->FillRandom("gaus",100000);
h->Draw();
myapp->Run();
return 0;
}
```

http://caeinfo.in2p3.fr/root/Formation/fr/Jour5/MyApp.C

Sous Unix/Linux/MacOsX

```
g++ MyApp.C -I$ROOTSYS/include `root-config --libs` `root-config --glibs`
a.out

g++ MyApp.C -DWITHRINT -I$ROOTSYS/include `root-config --libs` `root-config --glibs`
a.out
```