Introduction to ROOT

Prof. Silvia Masciocchi
dr. Federica Sozzi

Heidelberg University

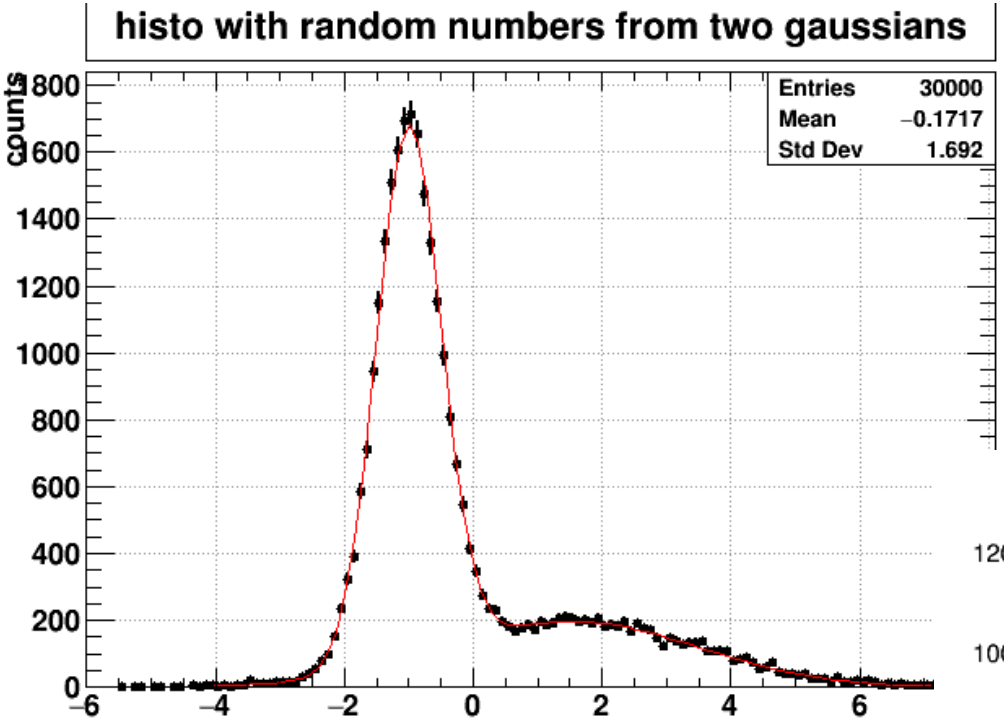
Day 3 – More into the deep

Based on the slides created by dr. Jens Wiechula, Frankfurt University

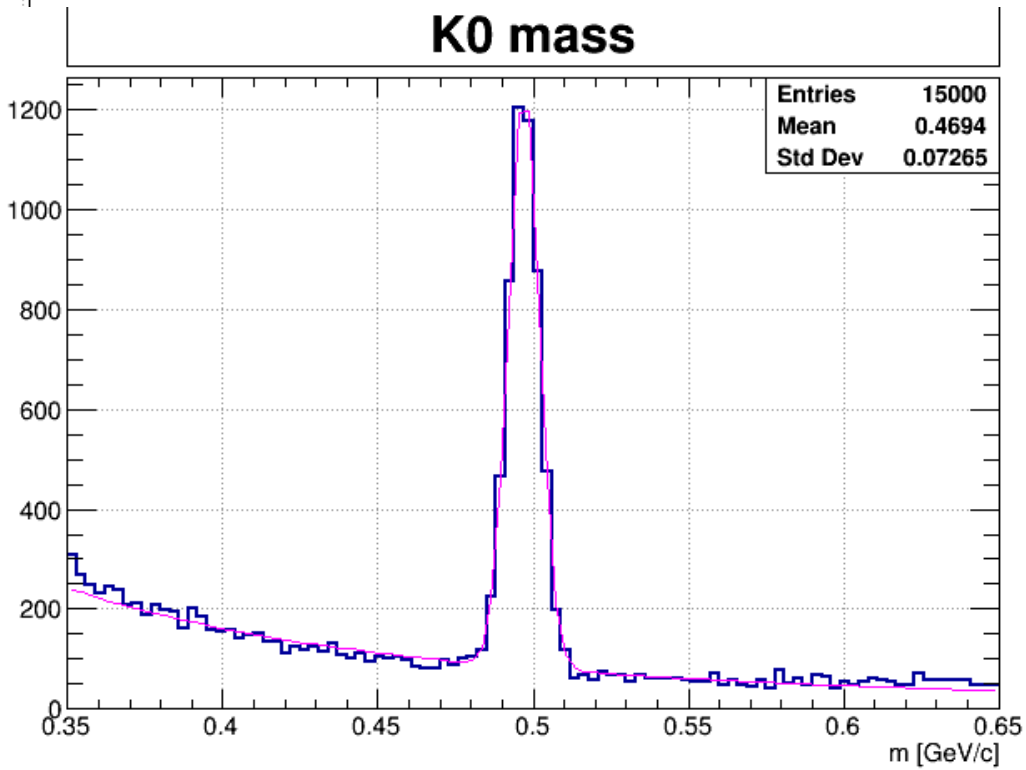
Additions to yesterday's lecture

Answers to some of the questions raised during the exercises

Fits



Two of the exercises from yesterday:
Fit using the sum of two Gaussians
or Gaussian + a function for a background



exercises/day02/FillRandomAndFit.C
exercises/day02/drawK0.C

Fits

One way to define the sum of two gaussians

```
// gaus(x) means that the indices of the parameters for that gaussian start at x.
// Since a gaussian has 3 parameters, the parameters of the second gaussian need to
// start at index 3
TF1 fFit("fFit", "gaus(0)+gaus(3)", -10, 10);

// Set some reasonable start parameters (deliberately not perfectly the ones
// used for random number generation!)
// We don't know the yield and there normalisations involved, so let's put just 100 here
Double_t yield1 = 100.;
Double_t yield2 = 100.;
Double_t mean1 = -0.9;
Double_t mean2 = 1.8;
Double_t sigma1 = 0.3;
Double_t sigma2 = 2.5;

fFit.SetParameter(0, yield1);
fFit.SetParameter(1, mean1);
fFit.SetParameter(2, sigma1);
fFit.SetParameter(3, yield2);
fFit.SetParameter(4, mean2);
fFit.SetParameter(5, sigma2);
```

Initial parameters are important.
If you choose them too far from the real values, the fit may not converge.

exercises/day02/FillRandomAndFit.C

Fits

Another way to define the sum of two functions

```
  //define the fit function as sum of two function:
 // gaus(x) means that the indices of the parameters for that gaussian start at x

 // Since a gaussian has 3 parameters, the parameters of the second function need
to start at index 3
  TF1 fFit("fFit", "gaus(0)+[3]*pow(x,[4])", 0.35,0.65);

 // Set some reasonable start parameters
 // in case the fit does not work, try to search the correct parameters for the t
wo functions individually:
 // Fit of the peak region with a gaussian
 // Fit  the whole region with the second function
 Double_t yield1 = 1000.;
 Double_t mean1 = 0.5;
 Double_t sigma1 = 0.005;
 Double_t exp0 = 7.;
 Double_t slope = -5.8;

 fFit.SetParameter(0, yield1);
 fFit.SetParameter(1, mean1);
 fFit.SetParameter(2, sigma1);
 fFit.SetParameter(3, exp0);
 fFit.SetParameter(4, slope);
```

One suggestion on how to choose reasonable initial values

The fits can be done via GUI or fitting in sub-ranges with the option “R”

See tutorial: $ROOTSYS/tutorials/fit/multifit.C

exercises/day02/drawK0.C

Fitting with RooFit

The RooFit library provides a toolkit for modeling the expected distribution of events in a physics analysis. Models can be used to perform unbinned maximum likelihood fits, produce plots, and generate "toy Monte Carlo" samples for various studies.

Mathematical concept		RooFit class
variable	x	`RooRealVar`
function	$f(x)$	`RooAbsReal`
PDF	$f(x)$	`RooAbsPdf`
space point	$\vec{x}$	`RooArgSet`
integral	$\int_{x_{min}}^{x_{max}} f(x)dx$	`RooRealIntegral`
list of space points		`RooAbsData`

https://root.cern.ch/download/doc/RooFit_Users_Manual_2.91-33.pdf

https://root.cern.ch/roofit-20-minutes

Reading data from a file (C++)

```
#include <iostream>
#include <fstream>
using namespace std;

void readDataFromFile(){

  //class to read an input file
   ifstream in;
   //open an existing file to read
   in.open("data.dat");

   Float_t x,y,z;
   while (1) {
     //insert the values of each line in the variables x y z
     in >> x >> y >> z;
     // when the file finishes, exit the loop
      if (!in.good()) break;
      //print the values
      cout<<x <<" "<<y<<" "<<z<<endl;
   }
   //close the file
   in.close();
}
```

example_code/day03/readDataFromFile.C

Writing data to a file (C++)

```
1  // basic file operations
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main () {
7    ofstream myfile;
8    myfile.open ("example.txt");
9    myfile << "Writing this to a file.\n";
10   myfile.close();
11   return 0;
12 }
```

```
[file example.txt]
Writing this to a file.
```

http://www.cplusplus.com/doc/tutorial/files/

Reading from a file with TGraph

Note that TGraphErrors have also a constructor that reads directly from a file.
The data file should contain columns with value_x, value_y, err_x, err_y

```
TGraphErrors::TGraphErrors ( const char *  filename,
                             const char *  format = "%lg %lg %lg %lg",
                             Option_t *    option = ""
                           )
```

And next week we will see that the same holds for TTrees

Outline of today

- Saving and reading of data
- The ROOT browser
- The base class of ROOT classes & casting
- Directories
- Collection classes
- Strings

Saving and reading of data

Introduction

- ROOT has a powerful mechanism to save any class into a file
- To store the data the class TFile is used
- The saved classes can easily be read back or graphical objects directly drawn
- All ROOT classes can be written to a file

Saving and reading of data

A first example

- Create a histogram
- Open a file
- Save the histogram
- List the contents of the file
- Close the file

GSI: wiechula@milamber: ~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day02

File Edit View Bookmarks Settings Help

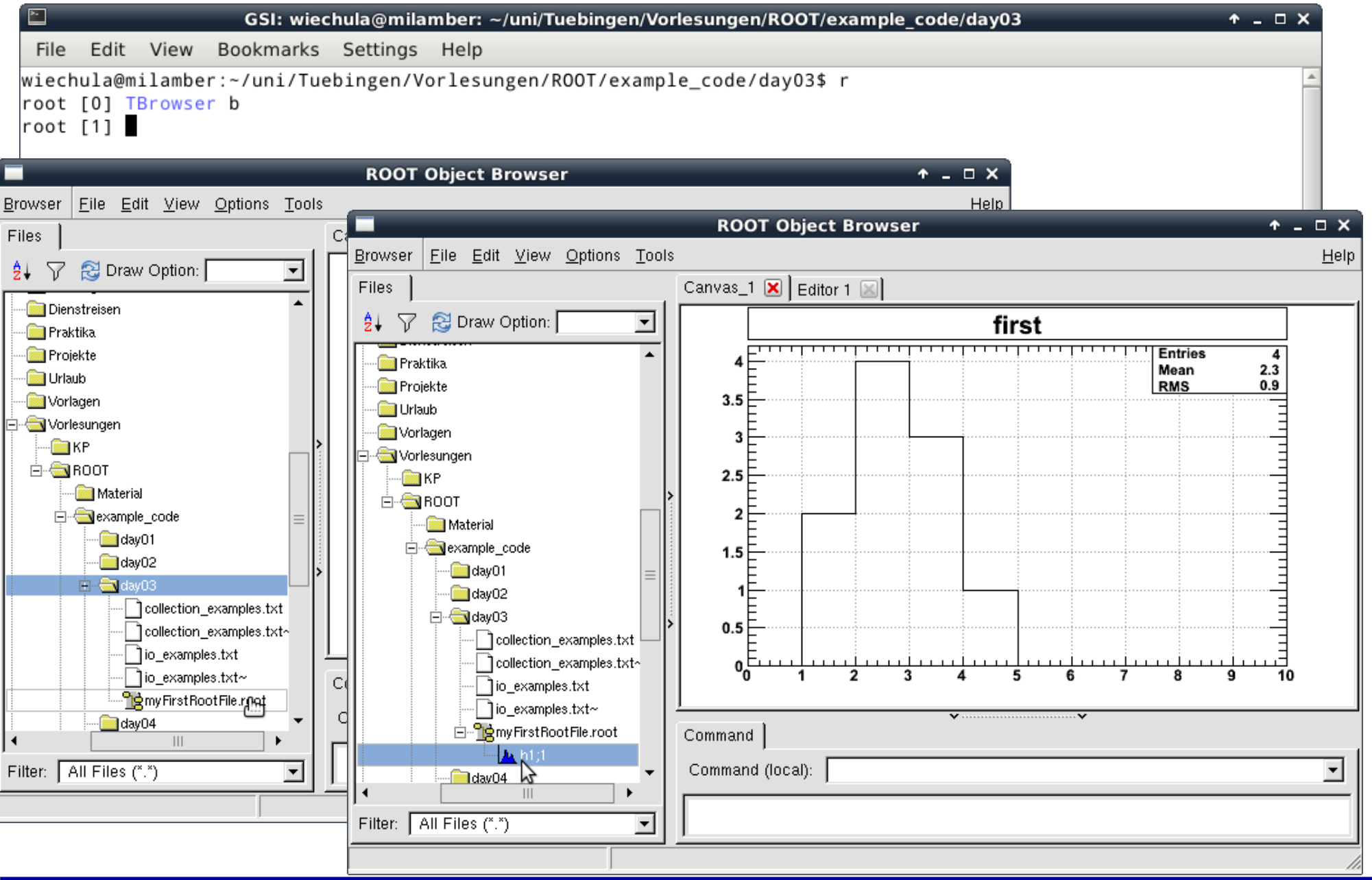
```
root [0]   TH1F *h1 = new TH1F("h1","first",10,0,10);
root [1]   h1->Fill(1,2);
root [2]   h1->Fill(2,4);
root [3]   h1->Fill(3,3);
root [4]   h1->Fill(4,1);
root [5]
root [5]   TFile f("myFirstRootFile.root","recreate");
root [6]   h1->Write();
root [7]   f.ls();
TFile**         myFirstRootFile.root
 TFile*         myFirstRootFile.root
  KEY: TH1F     h1;1    first
root [8]   f.Close();
root [9]
```

List the contents of the file

io_examples.txt

Saving and reading of data

A first example – using the browser



Saving and reading of data

TFile – constructor

TFile(const char* **fname**, Option_t* **option** = "", const char* **ftitle** = "", Int_t **compress** = 1)

fname name of the file

option

NEW or CREATE	create a new file and open it for writing, if the file already exists the file is not opened
RECREATE	create a new file, if the file already exists it will be overwritten
UPDATE	open an existing file for writing. if no file exists, it is created.
READ	open an existing file for reading (default)

ftitle title of the file (not needed)

compress compression level

Saving and reading of data

TFile – most important functions

Retrieve an object:

Get(const char* **name**)

name name of the object to retrieve

List the contents of the file

ls(Option_t* option = "")

option not important, see

http://root.cern.ch/root/html/TDirectoryFile.html#TDirectoryFile:ls

Close the file

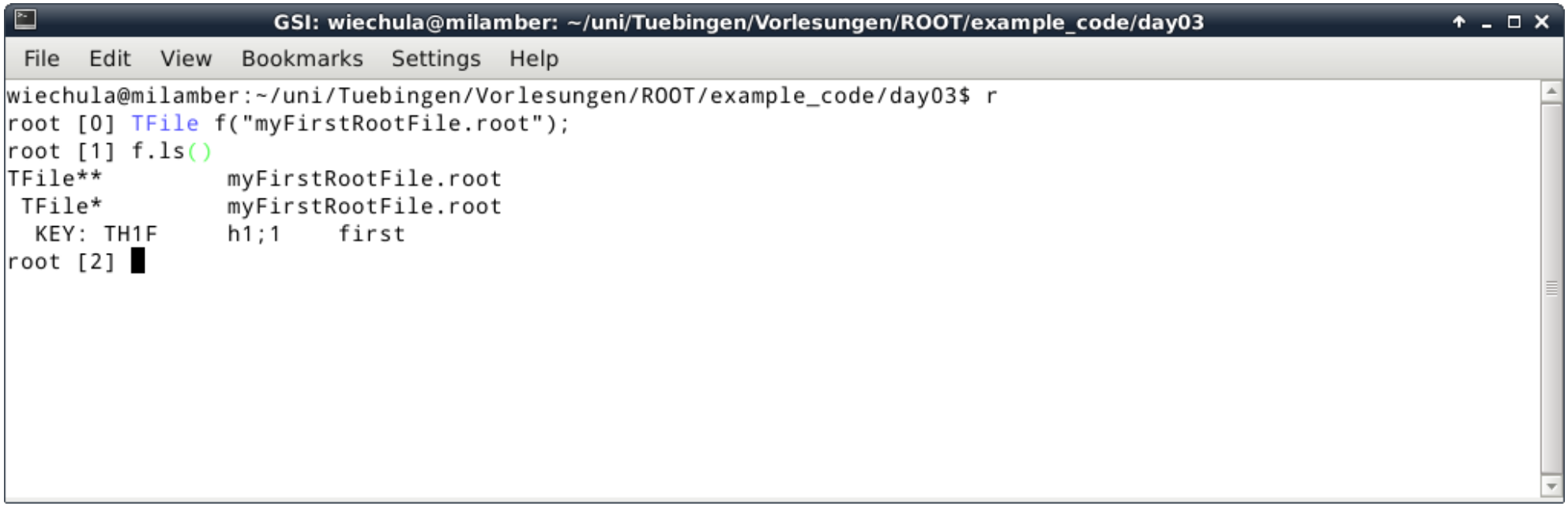
Close(Option_t* option = "")

option not important, see

http://root.cern.ch/root/html/TFile.html#TFile:Close

Saving and reading of data

Opening a root file



- Use TFile without second parameter (defaults to “read”)
- After this, the file is open and objects can be browsed

Saving and reading of data

Retrieving an object

GSI: wiechula@milamber: ~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day03

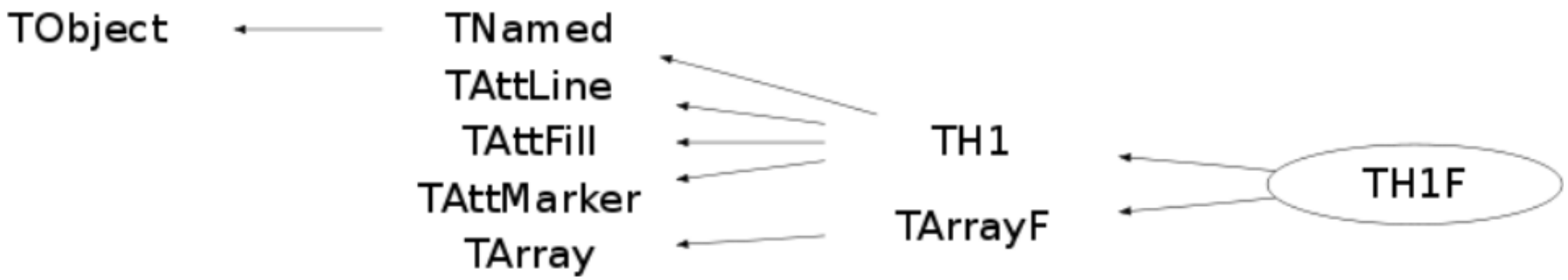
File Edit View Bookmarks Settings Help

```
wiechula@milamber:~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day03$ r
root [0] TFile f("myFirstRootFile.root");
root [1] f.ls()
TFile**         myFirstRootFile.root
 TFile*         myFirstRootFile.root
  KEY: TH1F     h1;1    first
root [2] TObject *o = f.Get("h1")
root [3] o->GetName()
(const char* 0x29e8739)"h1"
root [4] o->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [5]
```

- NOTE: since ANY ROOT object can be stored, TFile::Get(...) returns as type the ROOT base type TObject
- Like this also only the functionality of TObject can be used
- In order to convert this again to the correct object (shown in the file listing) one needs to cast the object to another type

Side remark

Casting 1



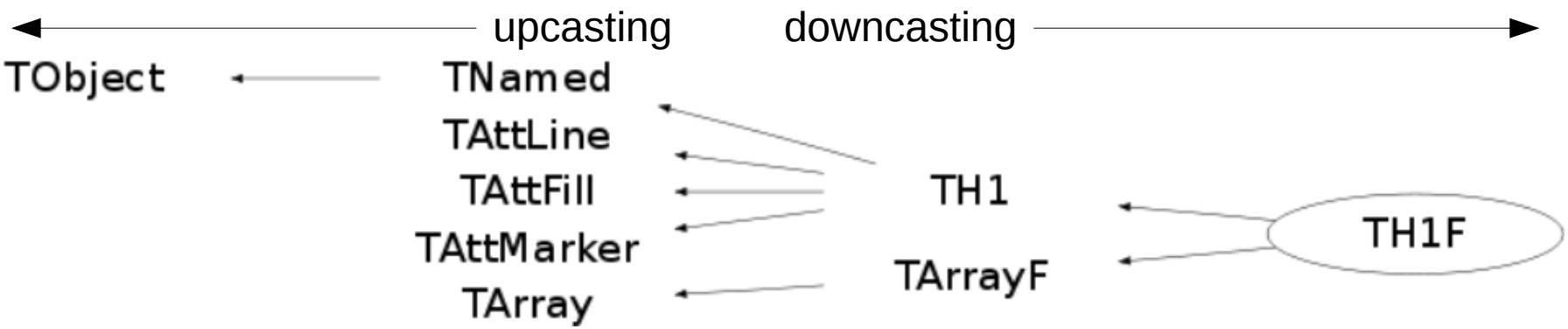

```
class TH1F : public TH1, public TArrayF
class TH1 : public TNamed, public TAttLine, public TAttFill, public TAttMarker
```

- Classes can 'inherit' from other classes
- This means they can extend or modify the possibilities of the inherited class and use their features
- e.g. TH1F inherits from TH1, which inherits from TNamed, which inherits from the ROOT base class TObject
- One can read this as: A TH1F is also a TH1 and is also a TNamed and is also a TObject
- BUT: A TObject is not necessarily a TH1F …

Side remark

Casting 2

- The objects can be 'converted' into each other using a method called 'casting'
- One can either cast to parent classes (upcasting) or to child classes (downcasting)



See also: http://www.cplusplus.com/doc/tutorial/typecasting/

Side remark

Casting 3

- Several methods exist for casting
- (type)object
  - e.g. TObject *o = (TObject*)myTH1F
  - This is the c-like type casting and for most purposes fine to be used, however discouraged, since it is not very explicit
- static_cast<type>(object)
  - e.g. TObject *o = static_cast<TObject*>(myTH1F)
  - More explicit cast, checks at least if a cast is possible at compiler time
- dynamic_cast<type>(object)
  - e.g. TObject *o = dynamic_cast<TObject*>(myTH1F)
  - This makes a full type check if the casting is allowed at run time, otherwise it returns a NULL pointer. More time consuming than the other casts
  - Only works in compiled code! In CINT it is like static_cast
- More info: http://stackoverflow.com/questions/28002/regular-cast-vs-static-cast-vs-dynamic-cast

Saving and reading of data

Retrieving an object and casting it

GSI: wiechula@milamber: ~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day03

File Edit View Bookmarks Settings Help

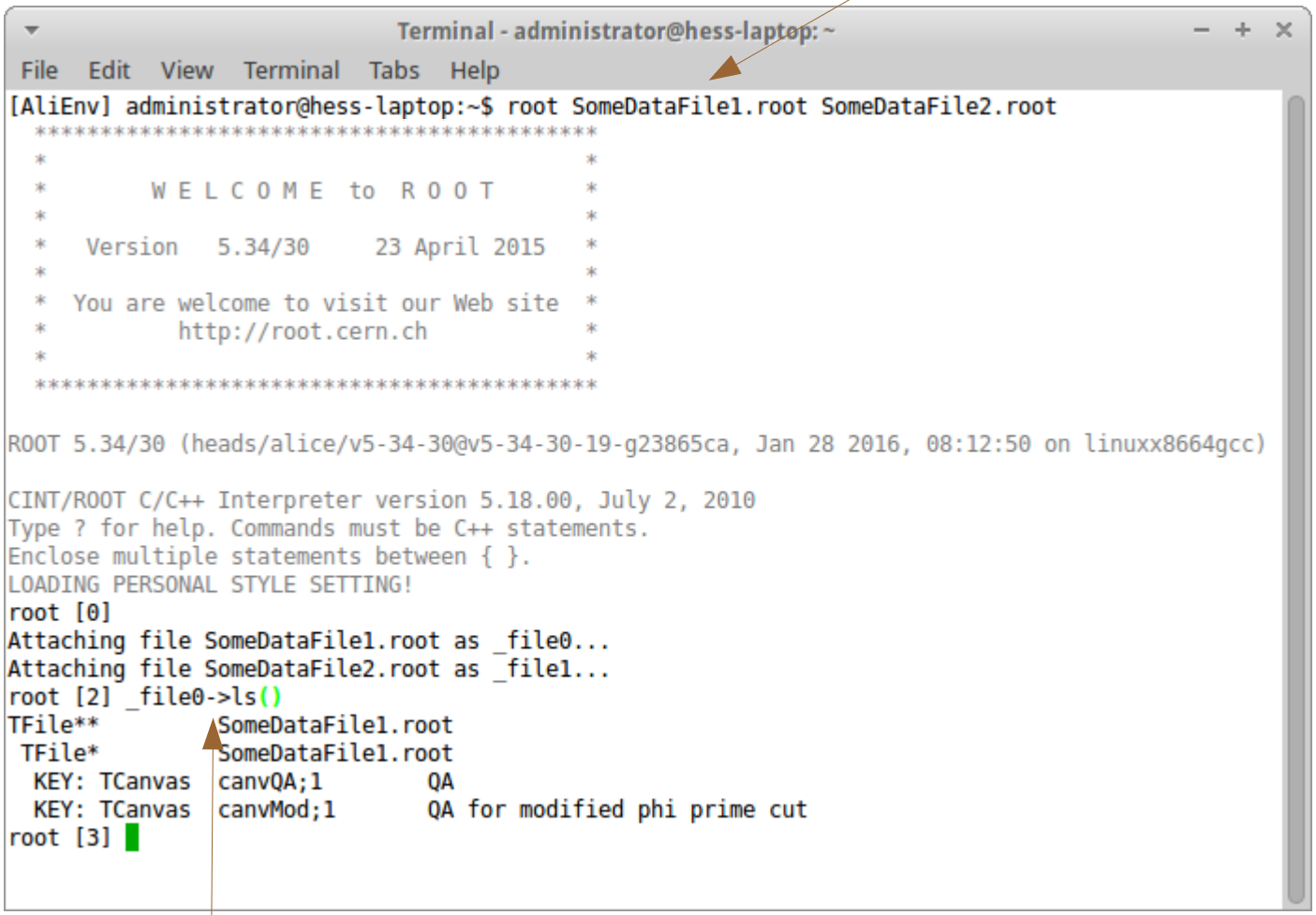
```
wiechula@milamber:~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day03$ r
root [0] TFile f("myFirstRootFile.root");
root [1] f.ls();
TFile**         myFirstRootFile.root
 TFile*         myFirstRootFile.root
  KEY: TH1F     h1;1    first
root [2] // we know we have save a histogram of type TH1F, so we can explicitly cast the TObject
root [3] //   to a TH1F
root [4] TH1F *h1 = (TH1F*)f.Get("h1");
root [5]
```

- When cast into the object itself, the full functionality of the object is available again

Reading of data

Using the shell

- Attach files to ROOT directly in the shell (just provide the relative or absolute pathname(s) to the file(s))



- Files are then available in CINT as _file0, _file1, ...

Remarks on saving objects

A few examples

- ROOT objects can also be directly written to file

```
TH1F *h1 = new TH1F("h1","h1",10,0,10);
h1->FillRandom("gaus");
h1->SaveAs("/tmp/histo.root")

TH1F *h2 = new TH1F("h2","h2",10,0,10);
h2->FillRandom("gaus"); h2->SetLineColor(kRed);
TCanvas c;
c.Divide(2);
c.cd(1);
h1->Draw();
c.cd(2);
h2->Draw();
 c.SaveAs("/tmp/savedCanvas.root");
```

Directories

Introduction

- ROOT has a concept of directories, similar to the file system
- When ROOT is started the base directory (gROOT) is selected
- A TFile is also a directory

- The global variable gDirectory always points to the current directory
- Histograms are by default associated to the current directory → This has a few implications

Directories

Remarks on histograms

```
1  // crate a histogram and list the
2  // contents of the current directory
3  TH1F *h1 = new TH1F("h1","h1",10,0,10);
4  gDirectory->ls();
5
6
7  // open a file first, create a histogram
8  // and list the contents
9  TFile f("tetFile.root","recreate");
10 TH1F *h2 = new TH1F("h2","h2",10,0,10);
11 gDirectory->ls();
12 f.Close();
13
14 // if you now try to draw h2 ROOT will crash
15 h2->Draw()
16
17 // the reason is that h2 is assiciated to the
18 // file and if the file is closed the histogram
19 // is deleted. However the pointer still exists
20 // but points to a position in memory where
21 // not histogram exists any longer
```

```
GSI: wiechula@milamber: ~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day
File  Edit  View  Bookmarks  Settings  Help
wiechula@milamber:~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day03$ r
root [0] TFile f("tetFile.root","recreate");
root [1] TH1F *h2 = new TH1F("h2","h2",10,0,10);
root [2] gDirectory->ls();
TFile**         tetFile.root
 TFile*         tetFile.root
  OBJ: TH1F     h2      h2 : 0 at: 0x1f810f0
root [3] f.Close();
root [4]
root [4] // if you now try to draw h2 ROOT will crash
root [5] h2->Draw()

 *** Break *** segmentation violation
```

Solution: remove the association to the file after this line by calling:
h2->SetDirectory(0x0);

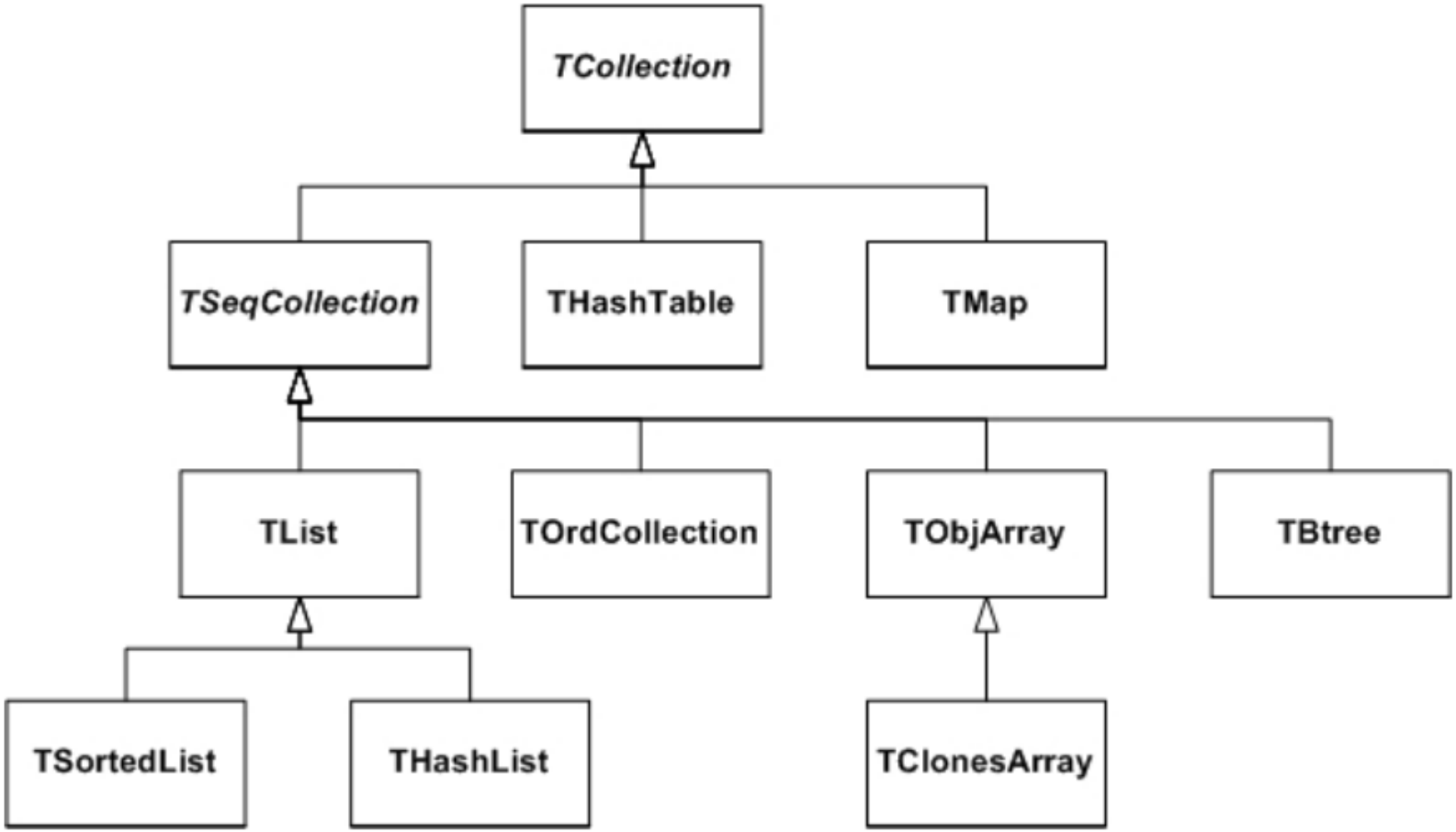
- Dealing with pointers can be tricky

directories_examples.txt

Collection classes

Introduction

- ROOT Provides several classes to group ROOT objects in one structure



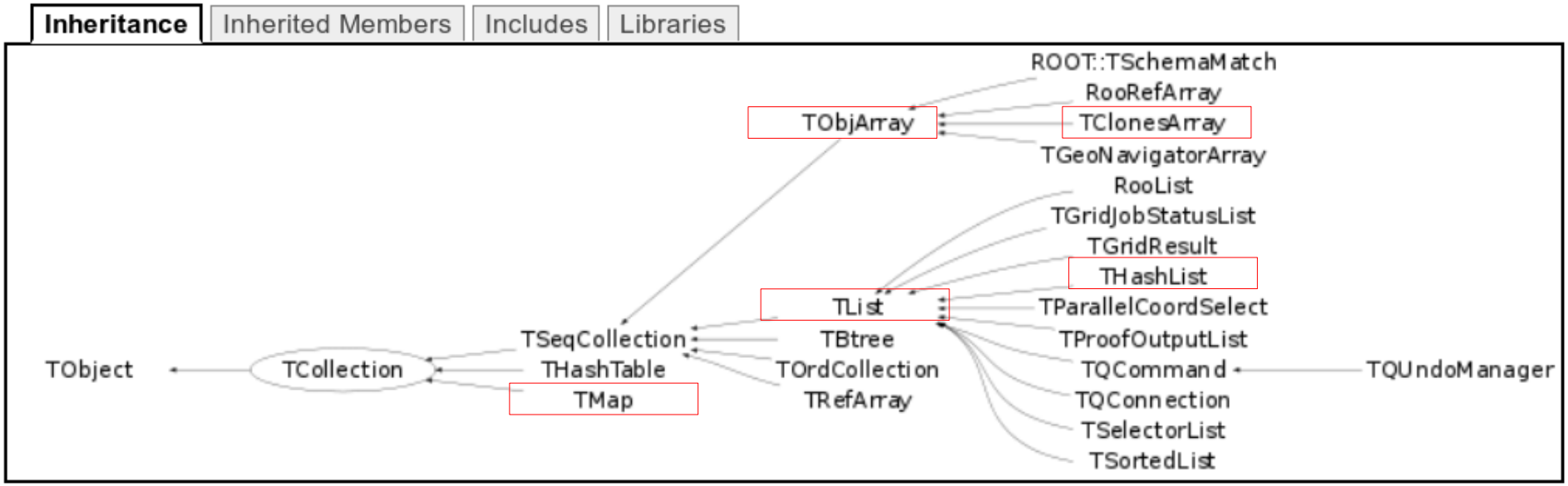
The inheritance hierarchy of the primary collection classes

Collection classes

Introduction

- ROOT Provides several classes to group ROOT objects in one structure
- They all inherit from TCollection

Class Charts http://root.cern.ch/root/html/TCollection.html



Collection classes

Introduction

- A collection is a group of related objects
- easier to manage a large number of items as a collection.
- Collections act as flexible alternatives to traditional data structures of computers science such as arrays, lists and trees
- Collections can only hold objects that inherit from TObject
- They return pointers to TObjects, that have to be cast back to the correct subclass
- Collections are dynamic; they can grow in size as required
- Collections themselves are descendants of TObject so can themselves be held in collections

Collection classes

TObjArray and TList

- TObjArray and TList are two of the most important collection classes and can store ANY object that inherits from TObject, also mixed types
- TObjArray is an array of objects with a certain size, objects don't need to be consecutive, so it can contain NULL pointers
- TList is a doubly linked list, an internal structure keeps references to the next and previous object in the list. It cannot contain NULL pointers

Collection classes

TObjArray and TList – most important functions

Adding an object to the collection

Add(TObject* **obj**)

Accessing objects in the collection

TObject* At(Int_t **idx**)

TObject* FindObject(const char* **name**)

TObject* FindObject(const TObject* **obj**)

Int_t IndexOf(const TObject* **obj**)

Number of elements in the collection

Int_t GetEntries()

Int_t GetEntriesFast() (only TObjArray)

Removing objects

TObject* Remove(TObject* **obj**)

TObject* RemoveAt(Int_t **idx**)

obj Any object that inherits from TObject

idx Position inside the collection

name name of the object

In case of TObjArray:

GetEntries() → Number of non-empty slots/number of objects in the array

GetEntriesFast() → Number of slots (including empty slots)

Collection classes

TObjArray and TList – example adding objects

```
1  // first example filling TObjArray or TList
2  // create a few histograms
3  TH1F *h1 = new TH1F("h1","h1",10,0,10);
4  TH1F *h2 = new TH1F("h2","h2",10,0,10);
5  TH1F *h3 = new TH1F("h3","h3",10,0,10);
6  TH1F *h4 = new TH1F("h4","h4",10,0,10);
7
8  // crate a list and a TObjArray
9  TList list;
10
11 list.Add(h1);
12 list.Add(h2);
13 list.Add(h3);
14 list.Add(h4);
15
16 TObjArray arr;
17 arr.Add(h1);
18 arr.Add(h2);
19 arr.Add(h3);
20 arr.Add(h4);
```

collection_examples.txt

Collection classes

TObjArray and TList – looping over the objects 1

- Several possibilities exist to loop over the objects in the collection
- The simplest is to loop over the number of entries, accessing them by their index

```
22  //
23  // simple for loop over TObjArray and TList
24  //
25  for (Int_t iObj=0; iObj<list.GetEntries(); ++iObj){
26    cout << "Object name in list at position " << iObj << ": " << list.At(iObj)->GetName() << endl;
27  }
28
29  // NOTE: For TObjArray use 'GetEntriesFast'! Be aware that you might retrieve a NULL pointer!
30  for (Int_t iObj=0; iObj<arr.GetEntriesFast(); ++iObj){
31    cout << "Object name in array at position      " << iObj << ": " << arr.At(iObj)->GetName() << endl;
32  }
```

- NOTE the difference between TList and TObjArray: GetEntries() ↔ GetEntriesFast()
- Reason: the array can have less objects than the maximum size (NULL in between)

collection_examples.txt

Collection classes

TObjArray and TList – difference

TObjArray

idx	name
0	h1
1	h2
2	h3
3	h4
4	h5

Remove(h3)

idx	name
0	h1
1	h2
2	0x0
3	h4
4	h5

GetEntries() = 5

GetEntriesFast() = 5

GetEntries() = 4

GetEntriesFast() = 5

Looping over GetEntries() after removing the object would only go up to 'h4'

TList

idx	name
0	h1
1	h2
2	h3
3	h4
4	h5

Remove(h3)

idx	name
0	h1
1	h2
2	h4
3	h5

GetEntries() = 5

GetEntries() = 4

Looping over GetEntries() works correctly, since Remove() removes the object from the linked list

Collection classes

TObjArray and TList – looping over the objects 2

- Another possibility is to use a so called 'iterator'
- Iterators provide a function to sequentially loop over the contents in a collection
- TIter can be used and provides the () operator to access the next object

```
48 TIter nextListItem(&list);
49 TIter nextArrItem(&arr);
50
51 TObject *o=0x0;
52
53 // () in this case is an operator that returns the next object
54 while ( (o=nextListItem()) ) {
55   cout << "Object name in list " << o->GetName() << endl;
56 }
57
58 while ( (o=nextArrItem()) ) {
59   cout << "Object name in list " << o->GetName() << endl;
60 }
```

collection_examples.txt

Collection classes

TObjArray and TList – looping over the objects 3

- If you find the notation with the () operator confusing, you can also use the function Next() of TIter instead – it does exactly the same:

```
63  // Equivalently, one can use the function Next() of the iterator
64  cout << endl << endl << "Alternative way of iterating...." << endl << endl;
65
66  TIter listIterator(&list);
67  TIter arrIterator(&arr);
68
69  o=0x0;
70
71  while ( (o=listIterator.Next()) ) {
72    cout << "Object name in list " << o->GetName() << endl;
73  }
74
75  while ( (o=arrIterator.Next()) ) {
76    cout << "Object name in list " << o->GetName() << endl;
77  }
```

collection_examples.txt

Collection classes

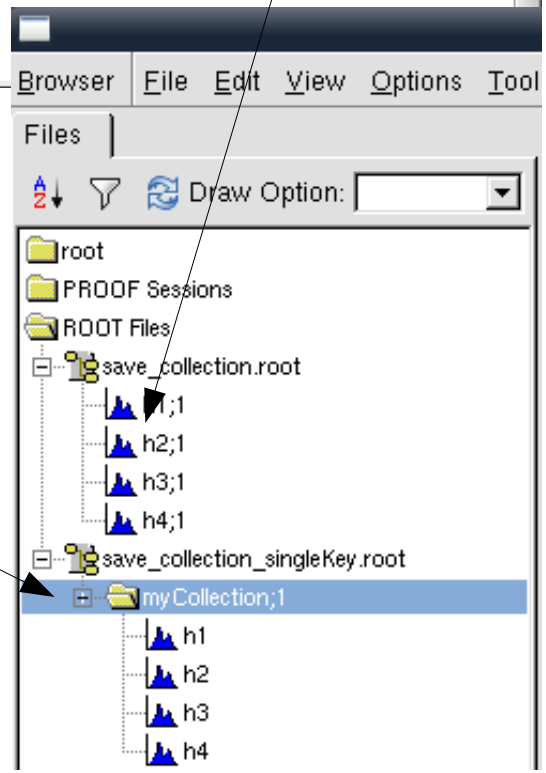
Saving collection classes

```
1  void save_collection()
2  {
3    // create a few histograms and add them
4    // to a collection
5    TH1F *h1 = new TH1F("h1","h1",10,0,10);
6    TH1F *h2 = new TH1F("h2","h2",10,0,10);
7    TH1F *h3 = new TH1F("h3","h3",10,0,10);
8    TH1F *h4 = new TH1F("h4","h4",10,0,10);
9  ..
10   // crate a list and a TObjArray
11   TList list;
12 ..
13   list.Add(h1);
14   list.Add(h2);
15   list.Add(h3);
16   list.Add(h4);
17
18   //open a file and save the collection
19   // look at the contents
20   TFile f("save_collection.root","recreate");
21   list.Write();
22   f.ls();
23   f.Close();
24
25   //open a file and save the collection
26   //now with the option kSingleKey
27   // look at the contents
28   TFile f2("save_collection.root","recreate");
29   list.Write("myCollection",TObject::kSingleKey);
30   f2.ls();
31   f2.Close();
32 }
```

```
GSI: wiechula@milamber: ~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day
File  Edit  View  Bookmarks  Settings  Help
wiechula@milamber:~/uni/Tuebingen/Vorlesungen/ROOT/example_code/day03$ r
root [0] .x save_collection.C
TFile**         save_collection.root
 TFile*         save_collection.root
  KEY: TH1F     h1;1    h1
  KEY: TH1F     h2;1    h2
  KEY: TH1F     h3;1    h3
  KEY: TH1F     h4;1    h4
TFile**         save_collection_singleKey.root
 TFile*         save_collection_singleKey.root
  KEY: TList    myCollection;1  Doubly linked list
root [1]
```

Without kSingleKey the contents of the collection are written directly

With kSingleKey the whole collection is written as one object.
In the browser it appears as a folder



Collection classes

Cleaning up

- If a TCollection object (or one from a derived class) is deleted, the objects in that collection are NOT deleted per default (exception: TclonesArray)
- In other words: the TCollections do not own the objects they hold for the very good reason that the same object could be a member of more than one collection
- This behaviour can be changed by calling SetOwner(kTRUE/kFALSE). If the collection is owner of its contents, they will be deleted/destructed when the collection is deleted or (depending on the collection type) Clear() is called.

Strings
Introduction

- Strings in ROOT are handled with the class TString
- In addition, a wrapper class that inherits from TObject exists: TObjString
- TString allows for very easy manipulation of strings and is recommended to be used over the c functions for string manipulation

Strings

A first example

```
root [0] // simple example: define two strings and add them
root [1]   TString s1("Hi");
root [2]   TString s2="over there";
root [3]   TString s3=s1+" "+s2+"!";
root [4]   cout<<s3<<endl;
Hi over there!
```

string_examples.txt

- String can be initialised using the constructor:

  TString(const char* s) s: default c-string
- Or using the assignment operator (like for numbers)
- In CINT: As for numbers simply entering a TString variable shows its contents

Strings

Most important functions

Access to internal c-string

const char* Data()

c-strings are often needed e.g. for function arguments

Adding text

Append(const char* **cs**)

Prepend(const char* **cs**)

Replacing

ReplaceAll(const char* **cs1**, const char* **cs2**)

… and many more functions

Parts of the string

char operator()(Ssiz_t **i**)

TSubString operator()(Ssiz_t **start**, Ssiz_t **len**)

Split string into sub-strings at a token

TObjArray* Tokenize(const TString& **delim**)

cs	A c-string → most functions are overloaded and can also take TString instead of a normal c-string
i	position in the string
start	start position for sub-string
len	length of sub-string
delim	tokens at which to split

Strings

The useful function “Form”

Since we are often dealing with c-strings, you you may find the function

char* Form(const char* **fmt**, …)

useful (with **fmt** being a printf style format descriptor)

```
1  TString path = "myPath/to/some/file";
2
3  for (Int_t fileIdx = 0; fileIdx < 5; ++fileIdx) {
4    TString fileName = Form("myFile%d.root", fileIdx);
5    TString filePathName = Form("%s/%s", path.Data(), fileName.Data());
6    printf("fileIdx %d: filePathName=\"%s\"\n\n", fileIdx, filePathName.Data());
7  }
```

Form_examples.txt

```
root [0] TString path = "myPath/to/some/file";
root [1]
root [1] for (Int_t fileIdx = 0; fileIdx < 5; ++fileIdx) {
end with '}', '@':abort >   TString fileName = Form("myFile%d.root", fileIdx);
end with '}', '@':abort >   TString filePathName = Form("%s/%s", path.Data(), fileName.Data());
end with '}', '@':abort >   printf("fileIdx %d: filePathName=\"%s\"\n\n", fileIdx, filePathName.Data());
end with '}', '@':abort > }
fileIdx 0: filePathName="myPath/to/some/file/myFile0.root"

fileIdx 1: filePathName="myPath/to/some/file/myFile1.root"

fileIdx 2: filePathName="myPath/to/some/file/myFile2.root"

fileIdx 3: filePathName="myPath/to/some/file/myFile3.root"

fileIdx 4: filePathName="myPath/to/some/file/myFile4.root"
```

printf example

The sintax followed by “Form” is the one of printf
This is the way in C to write to the stdout (in the previous lectures we have seen the C++ way, using cout)

```
1  /* printf example */
2  #include <stdio.h>
3
4  int main()
5  {
6     printf ("Characters: %c %c \n", 'a', 65);
7     printf ("Decimals: %d %ld\n", 1977, 650000L);
8     printf ("Preceding with blanks: %10d \n", 1977);
9     printf ("Preceding with zeros: %010d \n", 1977);
10    printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
11    printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
12    printf ("Width trick: %*d \n", 5, 10);
13    printf ("%s \n", "A string");
14    return 0;
15 }
```

http://www.cplusplus.com/reference/cstdio/printf/

What in practice you may need:
%s string
%d integer
%g float

If you don't like this “c-style” way of handling strings, don't worry.
You can also use the less-compact way as on the last slides to create the desired strings.

Strings

More examples

```
1  // simple example: define two strings and add them
2    TString s1("Hi");
3    TString s2="over there";
4    TString s3=s1+" "+s2+"!";
5
6
7
8  // more examples
9  //prepend and append
10 s1.Prepend("Why should I say ");
11 s1.Append(" To you?");
12 s1
13
14 //get one letter out of the string
15 // this is the 'y' of 'Why' in s1
16 TString oneLetter=s1(2);
17 oneLetter
18
19 //getting a substring starting at
20 // position 4 (the 's' in 'should')
21 // and being 6 letters long
22 TString aSubstring(s1(4,6));
23 aSubstring
24
25
26 //replace a string
27 s1.ReplaceAll("should I","should I not");
28 s1
```

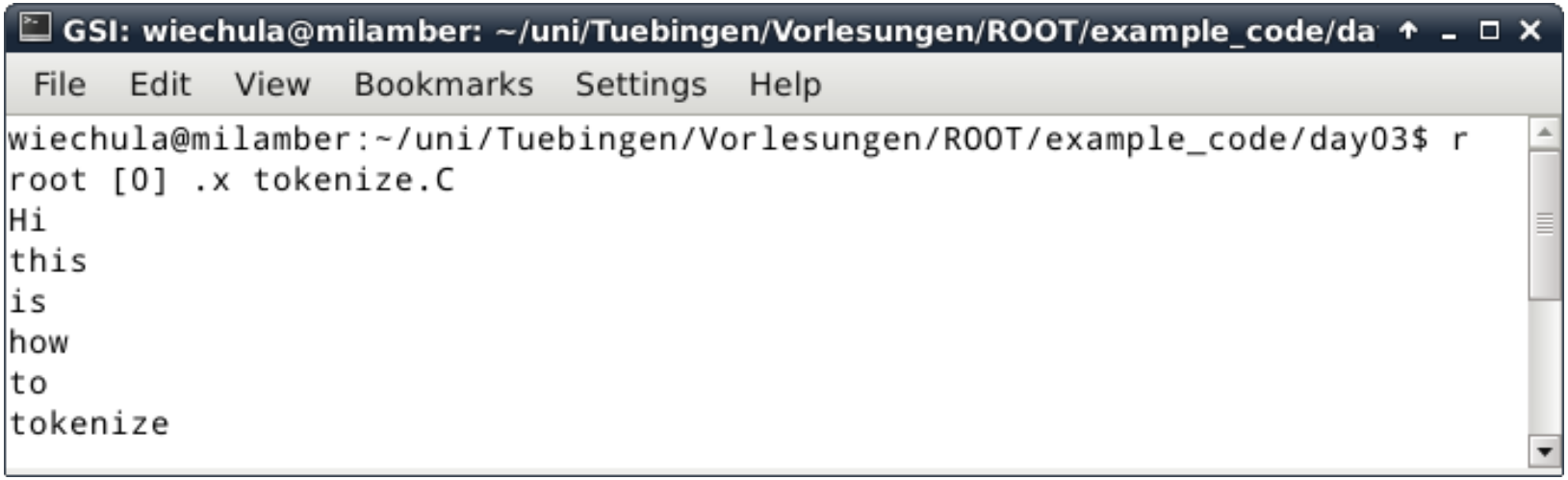
string_examples.txt

```
GSI: wiechula@milamber: ~/uni/Tuebingen/Vorlesungen/ROOT
File  Edit  View  Bookmarks  Settings  Help
wiechula@milamber:~/uni/Tuebingen/Vorlesungen/ROOT/example_
root [0]   TString s1("Hi");
root [1]   TString s2="over there";
root [2]   TString s3=s1+" "+s2+"!";
root [3]
root [3]
root [3]
root [3] // more examples
root [4] //prepend and append
root [5] s1.Prepend("Why should I say ");
(class TString)"Why should I say Hi"
root [6] s1.Append(" To you?");
(class TString)"Why should I say Hi To you?"
root [7] s1
(class TString)"Why should I say Hi To you?"
root [8]
root [8] //get one letter out of the string
root [9] // this is the 'y' of 'Why' in s1
root [10] TString oneLetter=s1(2);
root [11] oneLetter
(class TString)"y"
root [12]
root [12] //getting a substring starting at
root [13] // position 4 (the 's' in 'should')
root [14] // and being 6 letters long
root [15] TString aSubstring(s1(4,6));
root [16] aSubstring
(class TString)"should"
root [17]
root [17]
root [17] //replace a string
root [18] s1.ReplaceAll("should I","should I not");
(class TString)"Why should I not say Hi To you?"
root [19] s1
(class TString)"Why should I not say Hi To you?"
root [20]
```

Strings

Tokenize strings – an example

```
3  TString sToTokenize="Hi, this, is, how, to, tokenize";
4
5  //Tokenize the string at ',' and ' '
6  TObjArray *arr=sToTokenize.Tokenize(", ");
7
8  //the array not contains the tokenized words
9  //loop over them
10 TIter nextWord(arr);
11 TObject *o=0x0;
12
13 while ( (o=nextWord()) ){
14 |  cout << o->GetName() << endl;
15 }
```



Exercises: I/O writing and reading

- Create a histogram generated with a Gaussian distribution (mean=2,sigma=0.2) and write it in a file. Close the file

- Open the file and check the content with the TBrowser

- Create another histogram (e.g. Gaussian with mean 3 and sigma 0.2) and write it inside the same file (do not overwrite the previous one)

- Open the file, retrieve the histograms and draw them on the same canvas, using different line colors

- create a TLegend for the two histograms

Solution : exercises/day03/writingReading.C

Exercises: I/O + save canvas

- Open the file “example_code/day03/vertex.root”
- See the content either with TBrowser or via the root prompt
- Write a macro opening the file, getting the objects and drawing them in different pads of a TCanvas
- Save the TCanvas in a root file
- open the file and draw the TCanvas

Solution : exercises/day03/readVertex.C

Exercises: collection and strings

- Create several histograms and graphs. Fill the histograms in one collection class and the graphs in another. Save both to a root file using kSingleKey (save them with different names) – browse the file

  Solution: exercises/day03/collections.C

- Play around with strings and tokenisation
  - Example:

    Define a string as “/data/run2/histos/2016/09/15/”

    Retrieve the different part of the date from it (day, month, year) using Tokenize and print them on the stdout.

    Change the subdir “histos” in “graphs”

  Solution: exercises/day03/stringTest.C

Exercises: gStyle

- Draw a histogram and change the information printed in the statistics box via gStyle→SetOpt(…)

  For example add the overflow, underflow, the error of the mean

- Take one of your fit of yesterday and use gStyle→SetOptFit(...) to display a box with the fit results