Creating analysis script using MakeClass()

Anna Simon

December 2, 2015

- CIFs - now available at: https://cif.nd.edu/students/, open until Dec 13th.
- Final project due Dec 11th.
- Contact me regarding issues with the project BEFORE the due date.
- Do not skip the last class on Dec 9th.

The shortest ever intro to C++ classes

```
1  class Rectangle{
2      int width, height;
3    public:
4      Rectangle();
5      Rectangle(int,int);
6      int area();}
7  };
8
9  Rectangle::Rectangle(){
10   width = 5;
11   height = 5;
12 }
13
14 Rectangle::Rectangle(int a, int b){
15   width = a;
16   height = b;
17 }
18
19 Rectangle::area(){
20  return width*height;
21 }
```

```
1  int main(){
2    Rectangle rect(3,4);
3    Rectangle rectb;
4    cout << "rect area: " << rect.
         area() << endl;
5    cout << "rectb area: " << rectb.
         area() << endl;
6    return 0;
7  }
```

Creating your own analysis class

- open a file containing the tree you want to analyze

```
1 TFile *fIn = new TFile("run00166.root")
```

- run the MakeClass() function on the tree

```
1 t->MakeClass("analysis")
```

  This will create two files `analysis.C` and `analysis.h`

Inspect the content of the generated files.

How to use the analysis class

From the comments in `analysis.C`:

```
1 //    In a ROOT session, you can do:
2 //       Root > .L analysis.C   // Load the analysis class
3 //       Root > analysis t      // Create an analysis object "t"
4 //       Root > t.GetEntry(12); // Fill t data members with entry
      number 12
5 //       Root > t.Show();       // Show values of entry 12
6 //       Root > t.Show(16);     // Read and show values of entry 16
7 //       Root > t.Loop();       // Loop on all entries
```

- Load your analysis script:

```
1 .L analysis.C
```

- Create an object of type analysis:

```
1 analysis ana
```

- Execute the loop:

```
1 ana.Loop();
```

How to speed it up

In the Loop() function enable only the branches you need for analysis:

```
1     fChain->SetBranchStatus("*",0);  // disable all branches
2     fChain->SetBranchStatus("clover",1); // activate branch "clover"
```

The names of the branches you can figure out from analysis.h.

How to use it for any file

- Load the analysis script:

```
.L analysis.C
```

- Open the file you want to analyze:

```
TFile *fIn = new TFile("run00177.root");
```

- Create a pointer to the tree within that file that you want to analyze:

```
mytree = (TTree*) gDirectory->Get("t");
```

- Create an analysis object specifying the pointer as a constructor's argument (this will use the tree `mytree` is pointing to)

```
analysis ana(mytree)
```

- Execute the loop:

```
ana.Loop()
```

Analyze multiple files one-by-one

```
1  // load analysis script
2  .L analysisComments.C
3
4  //analyze the first file
5  TFile *_file0 = TFile::Open("run00172.root")
6  mytree = (TTree*)gDirectory->Get("t")
7  analysis ana(mytree)
8  ana->Loop()
9
10 //reset variables within the ROOT session (ana and mytree are
       destroyed)
11 gROOT->Reset()
12
13 //analyze the second file
14 TFile *_file0 = TFile::Open("run00166.root")
15 mytree = (TTree*)gDirectory->Get("t")
16 analysis ana(mytree)
17 ana->Loop()
```

Convert the script into a stand-alone code

Create main.C file:

```
#include <iostream>

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TTree.h>

#include "analysis.h"

using namespace std;

int main(){
    TFile *file = new TFile("run00166.root");

    TTree *mytree = (TTree*)gDirectory->Get("t");

    analysis ana(mytree);

    ana.Loop();

    return 0;
}
```

Convert the script into a stand-alone code

- Edit the following lines in Makefile:

```
1 FILES          = main.C analysis.C
2 PROGRAMS       = main
```

- compile and run!
- now you can:
  - edit main() so you can pass a run number from the command line
  - edit Loop() so that it takes that run number as an argument, so you can identify the output files by it
  - make your analysis code as complex as you wish!