EP578 Computing for Physicists

Topic 11

ROOT: NTuples & Trees
Fundamentals

Department of
Engineering Physics

University of Gaziantep

Course web page
www.gantep.edu.tr/~bingul/ep578

Nov 2011

Content

- Plain Text Data
- Ntuples
- Trees
- Trees in Analysis

Introduction

- For simple applications,
  we can use plain-text files such as:

```
 2   He    4.004
 3   Li    6.941
 4   Be    9.012
 5   B    10.811
 6   C    12.011
 7   N    14.007
 8   O    15.999
 9   F    18.999
11   Na   22.990
12   Mg   24.305
```

- Or

Atomic number	Name	Symbol	Group	Period	State at STP	Description	Atomic masses (u)	MP oC	BP oC
1	Hydrogen	H	1	1	Gas	Non-metal	1.00794	-259.34	-252.87
2	Helium	He	18	1	Gas	Noble-gas	4.002602	-272.226	-268.934
3	Lithium	Li	1	2	Solid	Alkali-metal	6.941	180.54	1342
4	Beryllium	Be	2	2	Solid	Alkaline-earth-metal	9.012182	1278	2970
5	Boron	B	13	2	Solid	Metalloid	10.811	-7.2	58.78
6	Carbon	C	14	2	Solid	Non-metal	12.0107	3652	4600
7	Nitrogen	N	15	2	Gas	Non-metal	14.0067	-209.86	-195.8
8	Oxygen	O	16	2	Gas	Non-metal	15.9994	-218.4	-182.962
9	Fluorine	F	17	2	Gas	Halogen	18.9984032	-219.62	-188.4
10	Neon	Ne	18	2	Gas	Noble-gas	20.1797	-248.67	-246.048
11	Sodium	Na	1	3	Solid	Alkali-metal	22.98976928	97.81	882.9
12	Magnesium	Mg	2	3	Solid	Alkaline-earth-metal	24.305	648.8	1090
13	Aluminum	Al	13	3	Solid	Metal	26.9815386	660.37	2467
14	Silicon	Si	14	3	Solid	Metalloid	28.0855	1410	2355

See: http://www1.gantep.edu.tr/~bingul/ep578/docs/atomic.txt

- In case you want to store large quantities of same-class objects, ROOT has designed the **`TNtuple`** and **`TTree`** classes specifically to save root files.

- The **`TTree`** class is optimized to reduce disk space and enhance access speed.

- A **`TNtuple`** is a **`TTree`** that is limited to only hold floating-point numbers.

- A **`TTree`** can hold all kind of data), such as objects or arrays in addition to all the simple types.

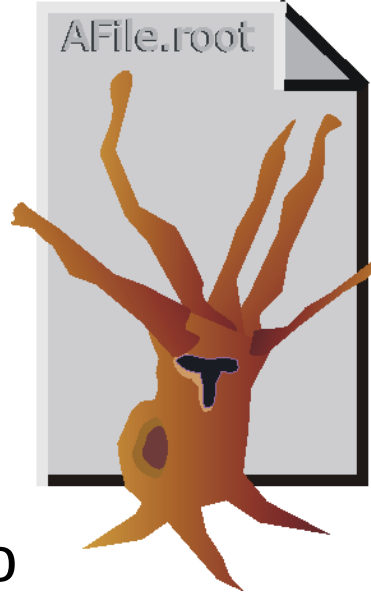
NTuples

- A **`TNtuple`** is a **`TTree`** that is limited to only hold floating-point numbers.

```
// *** WRITING NTUPLE ***
void ntuple1w()
{
   gROOT->Reset();
// create an ntuple file ntuple1.root
   TFile *f = new TFile("ntuple1.root","RECREATE");
   TNtuple *ntuple = new TNtuple("ntuple","just data","x:y:z");
   Float_t x,y,z;
// fill the ntuple
   for (int i=0; i<10000; i++) {
     x = gRandom->Uniform() + gRandom->Uniform();
         y = gRandom->Uniform(0,10);
         z = gRandom->Uniform(0,100);
     ntuple->Fill(x,y,z);
   }
// save the ntuple
   f->Write();
   f->Close();
}
```

```
// *** READING NTUPLE ***
void ntuple1r()
{
   gROOT->Reset();
// read the NTuple generated by ntuple1w
   TFile   *f  = new TFile("ntuple1.root");
   TNtuple *n1 = (TNtuple*) f->Get("ntuple");
   Float_t xx, yy, zz;
// branches
   n1->SetBranchAddress("x",&xx);
   n1->SetBranchAddress("y",&yy);
   n1->SetBranchAddress("z",&zz);
// create a histogram, read all entries and fill the histogram
   TH1F *hx   = new TH1F("histo"  ,"px distribution",100,0,2);
   Int_t nentries = (Int_t) n1->GetEntries();
   for (Int_t i=0; i<nentries; i++) {
      n1->GetEntry(i);
      hx->Fill(xx);
   }
   hx->Draw();
}
```

Trees



- When using a **`TTree`**, we fill its branch buffers with leaf data and the buffers are written to disk when it is full.

- Using **`TTree`**, the following items can be written to a ROOT file

  - variables of ordinary data types (**`int`**, **`double`**, etc),
  - objects from derived data types (class).
  - histogram objects
  - vectors
  - any object derived from **`TObject`**
  - . . .

```
// *** WRITING TREE ***
void tree1w(){
   gROOT->Reset();
// create a tree file tree1.root
   TFile *f  = new TFile("tree1.root","recreate");
   TTree *t1 = new TTree("t1","a simple Tree with simple variables");
   Float_t px, py, pz;
   Int_t ev;
// set branch addresses
   t1->Branch("px",&px,"px/F");
   t1->Branch("py",&py,"py/F");
   t1->Branch("pz",&pz,"pz/F");
   t1->Branch("ev",&ev,"ev/I");
// fill the tree
   cout << "Filling tree..." << endl;
   for (Int_t i=0; i<10000; i++) {
     px = gRandom->Gaus(0,2);
     py = gRandom->Gaus(0,3);
     pz = gRandom->Gaus(0,1);
     ev = i;
     t1->Fill();
   }
// save the Tree header
   t1->Write();
   cout << "done." << endl;
}
```

```
// *** READING TREE ***
void tree1r(){
   gROOT->Reset();
// read the Tree generated by tree1w
   TFile *f = new TFile("tree1.root");
   TTree *t1 = (TTree*)f->Get("t1");
   Float_t px, py, pz;
   Int_t ev;
// branches
   t1->SetBranchAddress("px",&px);
   t1->SetBranchAddress("py",&py);
   t1->SetBranchAddress("pz",&pz);
   t1->SetBranchAddress("ev",&ev);
// create two histograms, read all entries and fill the histograms
   TH1F *hpz   = new TH1F("hpz"  ,"pz distribution",100,-3,3);
   TH2F *hpxpy = new TH2F("hpxpy","py vs px",30,-8,8,30,-8,8);
   cout << "Reading data..." << endl;
   Int_t nentries = (Int_t) t1->GetEntries();
   for (Int_t i=0; i<nentries; i++) {
      t1->GetEntry(i);
      hpz->Fill(pz);
      hpxpy->Fill(px,py);
   }
   TCanvas *c1 = new TCanvas("c1","Plot",10,10,700,500);
   c1->Divide(2,1);
   c1->cd(1); hpz->Draw();
   c1->cd(2); hpxpy->Draw("lego");
}
```

```
// *** READING TREE ***
void tree2r()
{
   gROOT->Reset();
// read the Tree generated by tree1w
   TFile *f = new TFile("tree1.root");
   TTree *t1 = (TTree*)f->Get("t1");
   Float_t px, py, pz;
   Int_t ev;
// branches
   t1->SetBranchAddress("px",&px);
   t1->SetBranchAddress("py",&py);
   t1->SetBranchAddress("pz",&pz);
   t1->SetBranchAddress("ev",&ev);
// We do not close the file. We want to keep the generated histograms
// we open a browser and the TreeViewer
   if (gROOT->IsBatch()) return;
   cout << "Starting Browser..." << endl;
   new TBrowser ();
   t1->StartViewer();
}
```

Trees in Analysis

- You can analyse the root file in ROOT prompt

Terminal:

```
> root tree1.root
```

ROOT console:

```
root [0] TBrowser g
```

ROOT console:

```
root [0] t1->Print()

root [1] t1->Scan("px")
root [2] t1->Scan("px:py")
root [3] t1->Scan("px:py", "px>0 & py>0")

root [4] t1->Draw("pz")
root [5] t1->Draw("px:py")
root [6] t1->Draw("px:py", "px>0 & py>0")
```

Homeworks

1. Consider the ASCII file containg a data of 103 different periodic elements at:

   http://www1.gantep.edu.tr/~bingul/ep578/docs/atomic.txt

   a) Build a **`TTree`** from the file and generate another file called **`atomic.root`**.

   b) Using **`Scan()`** method

   i. List all properties of the Group 1 elements

   ii. List all properties of the Period 8 elements

   iii. List all properties of the Metals

   iv. List all properties of the elements whose boiling point are less than 100 oC.

   v. List all melting point of all solids.

   c) Reapet case b) in a ROOT macro file.