ROOT -Einführung
Statistische Methoden der Datenanalyse

Stan Lai, Florian Kiss

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

ROOT

- Softwarepacket zur Datenanalyse (http://root.cern.ch)
- Basiert auf der Programmiersprache C++
- Keine Vorkenntnisse von C++ vorausgesetzt für diese Übung
- heute: einfache Grundlagen, etwas kompliziertere Sachen bei der jeweiligen Übung nach Bedarf (aber auch: kein C++ Programmierkurs!)

- Terminalfenster starten
- darin ROOT starten: "root" und Enter eingeben
- Text ausscheiben lassen: cout<<"Test"<<endl;
- Jede Zeile/Befehl mit Semikolon abschliessen
- Logout: .q;

Variablen deklarieren

- Im folgenden:

  Eingabe an Kommandozeile — Bildschirmausgabe

- Neue ganze Zahl "x" deklarieren:

```
int x=5;
```

- Format: *Typ Variablenname = Wert;*
- Ausgeben lassen:

```
cout << "x = " << x << endl;
```

```
x = 5
```

- x ändern und ausgeben lassen:

```
x = 3;
cout << "x ist jetzt " << x << endl;
```

```
x ist jetzt 3
```

- Typumwandlung:

```
int pi_int = 3.14;
cout << "pi_int = " << pi_int << endl;
```

```
pi_int = 3
```

- Fliesskommazahlen:

```
float pi_float = 3.14;
cout << "pi_float = " << pi_float << endl;
```

```
pi_float = 3.14
```

Kommandozeile ⇒ Script

- Viel zu aufwendig, alles immer an der Kommandozeile einzugeben
- Stattdessen: ROOT-Kommandos in Textdatei (Script) schreiben und diese Datei mit ROOT ausführen
- Beispiel: Datei mit Namen test.C:

```
{
   int x = 5;
   int y = 8;
   int z = x + y;
   cout << "x + y " << z << endl;
}
```

- Achtung: Text muss mit { anfangen und mit } aufhören
- Zwei Möglichkeiten, um test.C mit ROOT auszuf":
  1. ROOT starten und direkt test.C ausführen: "root test.C"
  2. Falls ROOT schon läuft: ".x test.C"
- Texteditorprogramm nach Wahl, z.B. "emacs" (an Kommandozeile eingeben)
- Kommentarzeilen: mit // beginnen

if - Bedingung

- "if": wird benutzt, um ein oder mehrere Kommandos nur dann auszuführen, falls eine Bedingung wahr ist,: "Falls es regnet, nimm einen Regenschirm mit"

```
int x=8;
if ( x<10 ){
   cout<<"x kann man mit zwei
   Haenden abzählen"<<endl;
}
```

x kann man mit zwei Haenden abzählen

```
int x=12;
if ( x<10 ){
   cout<<"x kann man mit zwei
   Haenden abzählen"<<endl;
}
```

- Tipp: Anstatt von

```
if (Bedingung A) { etwas; }
if ( nicht Bedingung A) { was anderes; }
```

geht auch:

```
if (Bedingung A) { etwas; }
else { was anderes; }
```

“for”-Schleife

- Ein Beispiel: Man will alle ganzen Zahlen von 0 bis 10 ausgeben:

```
cout<< 0 << endl;
cout<< 1 << endl;
...
cout<< 10 << endl;
```

- Besser: “for”-Schleife: “mach dieses Kommando bitte N mal”:

```
for (int i=0; i<=10; i++){
   cout << i << endl;
}
```

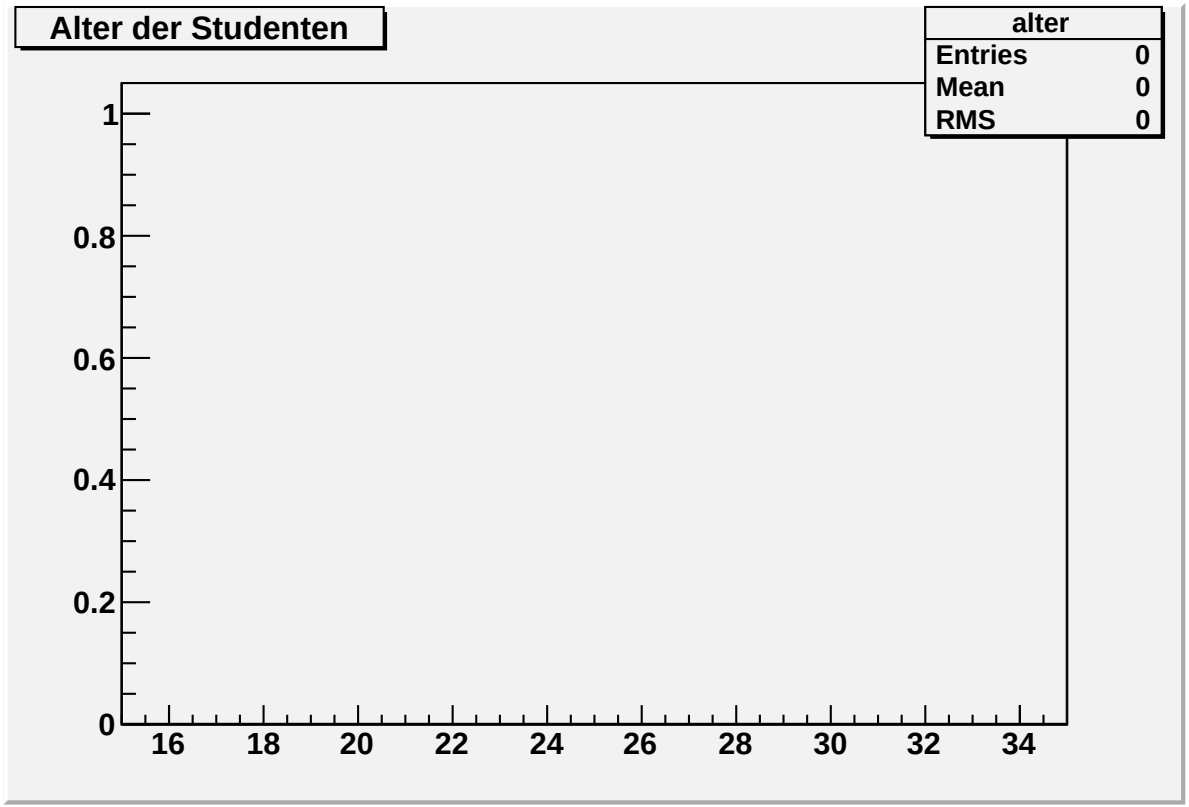
- Hinweis: “++” ist eine Abkürzung für “i=i+1”
- Andere Abkürzungen:
  - “x += y” ist gleich “x = x + y”
  - “x -= y” ist gleich “x = x - y”
  - “x *= y” ist gleich “x = x * y”
  - “x /= y” ist gleich “x = x / y”
- Test auf Identität: “x == y”
- Test auf Ungleichheit: “x!=y”

Histogramme

- Typ Name = Wert;
- int x = 6;
- Alternativ: Typ Name = Typ(Spezifikation von Eigenschaften)
- int x = int(6);
- ROOT-Histogramm: *T*yp ist "TH1F" (eigentlich: *K*lasse statt *T*yp
  TH1F hist = TH1F("Name","Titel", nBins, xLow, xHigh);
  - nBins: Anzahl der Bins im Histogramm – eine ganze Zahl (int)
  - xLow / xHigh: definieren die Intervallgrenzen der x-Achse
- Beispiel:

```
TH1F hist = TH1F("alter","Alter der Studenten", 20, 15, 35);
hist.Draw();
```

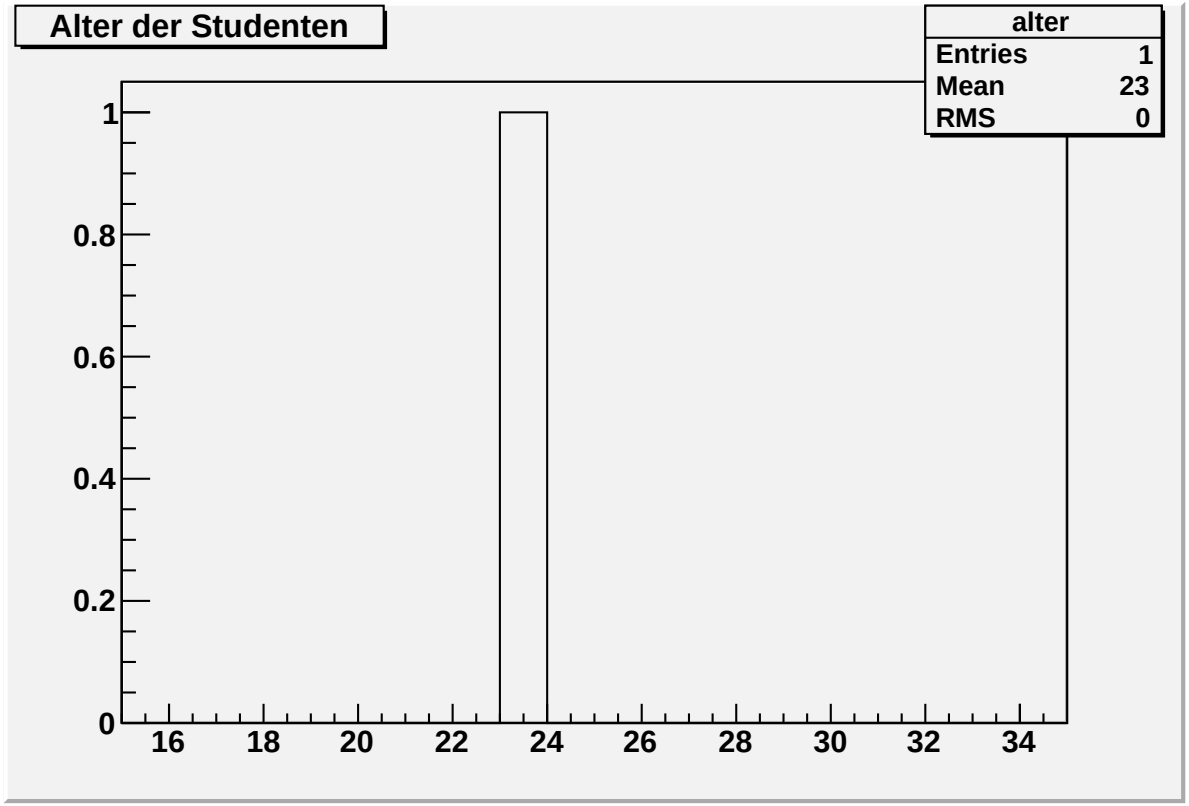
```
TH1F hist = TH1F("alter","Alter der Studenten", 20, 15, 35);
hist.Draw();
```



- "hist.Draw();" zeichnet Histogramm

Histogramme

```
TH1F hist = TH1F("alter","Alter der Studenten", 20, 15, 35);
hist.Fill(23);
hist.Draw();
```



- "hist.Fill(23);" fügt dem Histogramm einen Eintrag im Bin, das 23 einthält zu

Histogramme

- “Draw” und “Fill” werden “Mitgliedfunktionen der Histogrammklasse TH1F” genannt
- weitere Beispiele für nützliche Mitgliedfunktionen von Histogrammen:
  - “float x = hist.GetBinContent(N)”: Wert von “x” wird auf die Anzahl der Einträge im Bin Nummer N(=1,2,3,...) gesetzt
  - “float x = hist.Integral()”: Wert von “x” wird auf Anzahl der Einträge des Histogramms gesetzt
  - “hist.Scale(x)”: Multipliziert alle Einträge (Anzahlen) des Histogramms mit dem Wert von “x”
  - “int nBins = hist.GetNbinsX()”: Wert von “nBins” wird auf Anzahl der Bins des Histogramms gesetzt

“Zufallszahlen”

- ROOT kann Zufallszahlen generieren durch die Klasse TRandom3 und die Mitgliedfunktionen:
  - Integer(int max): gibt eine zufällige ganze Zahl zwischen 0 und (max-1)
  - Uniform(): gibt eine zufällige Fliesskommazahl zwischen 0 und 1
- Beispiel:

```
TRandom3 rnd = TRandom3();
for (int i=0; i<10; i++){ cout << rnd.Integer(10) << endl;}
```

4, 8, 3, 9, 7, 0, 8, 3, 0, 0

- Wenn man ROOT neu startet und das Beispiel wiederholt, kriegt man die gleichen 10 Zahlen nochmal
- Um 10 andere Zahlen zu bekommen, muss der sogenannte “Seed” geändert werden durch die Funktion SetSeed(int seed).

```
TRandom3 rnd = TRandom3();
rnd.SetSeed(12) for (int i=0; i<10; i++){ cout <<
rnd.Integer(10) << endl;}
```

1, 9, 3, 9, 1, 3, 0, 0, 8, 6