Wstęp do ROOT

ROOT: obiektowe środowisko programistyczne do analizy danych, oparte na C++

www : root.cern

Mody pracy

- Interaktywnie (sesja w terminalu)
  - polecenia C++ interpretowane on-line
  - Makra (interpretowane lub kompilowane), dwa możliwe mody:
    - uproszczony: ciąg poleceń
    - funkcje; wymóg większej zgodności ze standardem C++
- W kompilowalnym kodzie C++ : biblioteki Root'owskie

Zawiera

- Okna graficzne, histogramy, wykresy punktowe (z niepewnościami)
- Funkcje matematyczne (w tym: f. specjalne) : rysowanie, obliczanie, losowanie z rozkładu
- Dopasowywanie funkcji do rozkładów,
- Bazy danych ("drzewa"). Filtrowanie danych ("cięcia").
- Liczby pseudolosowe
- Kolekcje obiektów, I/O z zapisem obiektów
- Metody numeryczne. Analiza widm. Bazy typu DataFrame. Uczenie maszynowe. GUI.


krzysztof.piasecki@fuw.edu.pl

Przewodniki i pomoc

- Spis pomocy: root.cern/get_started/
- Podręcznik: root.cern/manual/
- Przewodnik na start root.cern/primer/
- Slajdy: indico.cern.ch/event/395198/attachments/791523/1084984/ROOT_Summer_Student_Tutorial_2015.pdf
- Forum: root-forum.cern.ch
- Dokumentacja: http://root.cern/doc/master

Uwaga: w niniejszym skrypcie nazwy klas zawierają linki do pomocy na stronie ROOT'a.

Instalacja

- Download: root.cern/install/#download-a-pre-compiled-binary-distribution
  źródła lub binaria dla: Linux, Windows, Mac
- Linux: do pliku logowania (`~/.bash_login` lub `~/.bashrc`) dodać linijkę:

```
. [sciezka_ROOTa]/bin/thisroot.sh
```

- Skok kwantowy:

Wersje ≤ 5.34	vs	Wersje ≥ 6.00
Interpreter "`cint`"		Interpreter "`cling`"
Tolerancja składni		Rygoryzm składni
Standard ~ c++98		Standard ~ c++11

MOD I: Sesja interaktywna ROOT jako kalkulator i interpreter poleceń C++

- Wywołanie:
  - `root`
  - `root -l` (bez okienka powitalnego)
  - `root -b` (bez wyświetlania grafiki, za to szybciej)

- W sesji.
  - Wyjście: `.q`
  - Polecenie w shellu: `.![polecenie]`
  - Wykonanie makra: `.x`
  - Wymuszenie wyjścia: `.qqqqqqq`

```
root [0] sqrt (1.23)
(const double)1.10905365064094164e+00
root [1] double x = pow (sin(0.5),2.) + pow (cos(0.5),2.)
root [2] x
(double)1.00000000000000000e+00
root [3] cout << x << endl
1
(class ostream)139768533438272
```

Typy zmiennych

① Typy C++ :

```
int, double, char tekst[100], string napis,
vector<double> vec, double* d, int& i, ...
```

② Wewnętrzne typy ROOT, dublujące typy C++ :

```
Int_t , Float_t , Double_t, Char_t, Bool_t
```

Motywacja: uniezależnienie kodu od rodzaju maszyny.

Lista takich typów:
root.cern/root/htmldoc/guides/users-guide/ROOTUsersGuide.html#machine-independent-types

TMath — Klasa matematyczna (Uwaga: w nazwach klas linki do pomocy)

- Funkcje (`TMath::Sqrt(x) , Power, SinH, Exp, Gaus, Factorial, ...`)
- Stałe matematyczne i fizyczne (`TMath::Pi(), E, RadToDeg, DegToRad, Hbar, K`)
- Operacje (`TMath::Abs(x) , Min, Max, ...`)
- Funkcje specjalne (`TMath::BesselI (x) , BesselJ/K/Y , Erf, ...`)

```
root [0] TMath::Power (TMath::Pi() , 1./3.)
(Double_t)1.4645918875615231e+00
```

Uwaga: w przestrzeni nazw `ROOT::Math` znajdują się kolejne funkcje i algorytmy.

Autouzupełnianie – inteligentna pomoc podręczna

```
root [1] TMath::Pi
Pi
PiOver2
PiOver4
root [1] TMath::Pi(

Double_t Pi()
root [1] TMath::Power(

LongDouble_t Power(LongDouble_t x, LongDouble_t y)
LongDouble_t Power(LongDouble_t x, Long64_t y)
LongDouble_t Power(Long64_t x, Long64_t y)
Double_t Power(Double_t x, Double_t y)
Double_t Power(Double_t x, Int_t y)
```

TCanvas — Okno graficzne

```
root[0] new TCanvas
(class TCanvas*)0x2c1e5e0
root[1] c1->Set <Tab>

root[1] c1->SetTitle ("HelloCanvas")
root[2] c1->GetTitle ()
(const char* 0x1557339)"HelloCanvas"
root[3] c1->ls ()
root[4] c1->Close()
root[5] TCanvas c2
root[6] c2.GetName()
(const char* 0x16632b1)"c1_n2"
root[7] TCanvas c3 ( <Tab>
```

← stworzenie “na szybko” okna graficznego o generycznych własnościach

← Uwaga: automatyczne nadanie nazwy “c1”

← Tworzymy nowe okno. Wcześniej był wskaźnik. Teraz – obiekt.

← Tytuł inny niż nazwa zmiennej!

Wielość konstruktorów

```
TCanvas TCanvas(Bool_t build = kTRUE)
TCanvas TCanvas(const char* name, const char* title = "", Int_t form = 1)
TCanvas TCanvas(const char* name, const char* title, Int_t ww, Int_t wh)
TCanvas TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy,
Int_t ww, Int_t wh)
TCanvas TCanvas(const char* name, Int_t ww, Int_t wh, Int_t winid)
root[7] TCanvas c3 ("c3canvas", "My canvas", 600, 400);
```

Nazwa własna obiektu (w ujęciu C++).

Identyfikator (Name) (równoległe nazewnictwo ROOTa).

Wyświetlany tytuł (zwykły c-string)

```
root[8] c3Canvas
(class TCanvas*)0x16964d0
root[9] TCanvas* c4 = new TCanvas ("c4canv", "2nd Canvas", 600, 400);
```

Name jako identyfikator lub zamiennik obiektu

Dynamiczna alokacja (używamy wskaźnika do obiektu)

TFn n = {1, 2, 3} Funkcje

```
root[0] TF1 f1 ("f1", "sin(x)/x", 0. , 10. );
root[1] f1.Draw ()
root[2] f1.SetRange (-10. , 10.)
root[3] f1.Draw ()
root[4] f1.Eval (1.)    lub   f1(1.)
root[5] f1.Integral ( 0. , TMath::Pi() )
root[6] f1.GetMinimum ( 1e-10 , 5.)
```

← funkcja o zadanym wzorze i zakresie

```
root[7] TF1 fb ("fb", "[0] * sin([1]*x)/x", 0., 10.);
root[8] fb.SetParameter (0, 0.5);
root[9] fb.SetParameter (1, 2. );
root[10] fb.Draw ("same")

root[11] f1.SetLineColor (2);
```

← funkcja uzależniona od parametrów

Numeracja kolorów w palecie generycznej

```
root[12] f1.SetLineWidth (2);

root[13] f1.SetLineStyle (2);

root[14] f1.Draw ("same");
```

Numeracja stylów linii

```
root[15] TF2 f3 ("f3", "exp(-(x-0.5)*(x-0.5)/0.05-(y-0.5)*(y-0.5)/0.05)");
root[16] f3.Draw ()
root[17] f3.Draw ("lego2")
root[18] f3.Draw ("colz")
```

← Opcje graficzne, np: “surf”, “surf2”, “cont”

root.cern/doc/master/classTHistPainter.html#HP01

(*Uwaga*: Są jeszcze inne sposoby tworzenia funkcji, możliwe w makrach C++.)

Zapisanie okna graficznego na pliku:

```
root[1] c1->Print ("picture.ext");
```

ext = {gif, jpg, pdf, png, ps, svg, root, tex, tiff, xml, xpm, C}

Uwaga: stratne -vs- bezstratne formaty zapisu grafiki

```
root[2] c1->Clear ();
```

TGraph Grafy danych

(i pokrewne: TGraphErrors / TGraphAsymmErrors / TGraphBentErrors)

tgraph.dat

```
0.0 0.01
0.1 0.98
0.2 2.03
0.3 2.99
0.4 4.07
0.5 5.01
```

W konsoli: `wget www.fuw.edu.pl/~kpias/ctnp/tgraph.dat`

```
root[1] TGraph g ("tgraph.dat")
root[2] g.Draw ("AP")

root[3] g.SetMarkerSize (0.8)
root[4] g.SetMarkerStyle (20)
root[5] g.Draw ("AP")
```

Style markerów

Można preselekcjonować kolumny danych:

```
wget www.fuw.edu.pl/~kpias/ctnp/tgraph2.dat

root[1] TGraph g ("tgraph2.dat", "%*s %lg %*lg %lg");
root[2] g.Draw ("AP");
```

`%lg` : wczytaj kolumnę jako double
`%*lg`: kolumna jest w double; pomiń ją
`%*s` : kolumna jest słowem; pomiń ją

```
root[3] TGraph gr2(
(…)
TGraph TGraph (Int_t n, const Double_t* x, const Double_t* y)
TGraph TGraph (Int_t n, const Float_t* x, const Float_t* y)
TGraph TGraph (Int_t n, const Int_t* x, const Int_t* y)
(…)

root[6] Double_t x[] = {0.05, 0.95, 1.95, 2.05, 3.05, 3.95};
root[7] Double_t y[] = {1.00, 1.11, 1.29, 1.41, 1.52, 1.59};
root[8] TGraph gr2 (6, x, y);
root[9] gr2.SetMarkerSize (0.8)
root[10] gr2.SetMarkerStyle (24)
root[11] gr2.Draw ("sameP")
```

TRandomN , *N* = { 1, 2, 3 } Liczby pseudolosowe

- Wg dokumentacji zaleca się używanie `TRandom3` . Czas wywołania ~ 45 ns. Okres ~ 10^{6000} .

```
root[1] TRandom3 r;
root[2] r.SetSeed ();          ← ustawianie pseudolosowego ziarna
```

- TRandomN posiadają predefiniowane rozkłady, m.in. :

```
Binomial (ntot, prob)          BreitWigner (mean, gamma)
Exp (tau)                      Integer (imax)
Landau (mean, sigma)           Gaus (mean, sigma)
Rndm ()   ← zwraca double ∈ [0, 1)   Poisson (mean)
```

```
root[3] r.Rndm ()
(Double_t) 9.997417e-01
root[4] r.Gaus (15.3, 0.02)
(Double_t) 1.531998e+01
```

- Ponadto, można losować liczby z rozkładu zadanego przez nas w formie funkcji `TF1`. Np. :

```
root[5] TF1 fun1 ( "fun1", "x*x*exp(-x/0.5)" , 0., 5.) ;
root[6] fun1.GetRandom ()
(Double_t) 6.916067e-01
```

- Można też losować z zadanego histogramu (histogramami zajmiemy się wkrótce).

TVectorN , $N = \{2, 3\}$ Wektory 2-3 wymiarowe

```
root[1] TVector3 v1 (1, 2, 3) , v2, v3;
root[2] v2.SetXYZ (-1,-2,-3);

        v1.Mag()  Mag2()   Theta()  CosTheta()  Phi()  Perp()   ← podst. własności

root[3] v3 = -3. * (v1 + v2);  v3 -= 2. * v1;  v3.Print()
root[4] v1.Cross(v2) .Print();                  ← iloczyn wektorowy
root[5] v1.Dot ( v3.Orthogonal () )             ← iloczyn skalarny;  wektor ⊥

        v1.Angle (v2);    v1.RotateX/Y/Z (kąt)  ;  v3.Rotate (kąt, v2)
```

TLorentzVector Czterowektor

Posiada 4 zmienne, których można używać albo jako [X, Y, Z, T] , albo [P_x, P_y, P_z, E] . Uwaga na kolejność! Jest zaimplementowany jako TVector3 ⊕ double.

```
root[1] TLorentzVector L (1, 2, 3, 4); cout << L.T() << endl;

        L.Pt() P()          ←  √(P_x² + P_y²) ,  |P⃗|
        L.M()               ←  ±√(interwał czasoprzestrzenny) / ±energia dostępna / ±masa niezmiennicza
        L.Beta() Gamma()    ←  β = ±|P⃗|/E,  γ = 1/√(1 − β²)

root[2] TLorentzVector v4piplus (0., 0., 1., sqrt(1*1 + 0.1395*0.1395) );
root[3] v4piplus.Rapidity();    ←  y = 0.5 · ln [(E − p_z)/(E + p_z)]

root[4] TVector3 beta (0., 0., 0.7);
root[5] v4piplus.Boost (beta);  ←  Transformacja Lorentza o parametr Lorentza "beta"
```

Histogramy

THdt, *d* = {1, 2, 3}
t = {C, S, I, F, D}

Typ zmiennej przechowywującej zawartość słupka ("*binu*")

Wymiar histogramu

Typ	C	S	I	F	D
Max. zaw. binu	$2^8 - 1$	$2^{16} - 1$	$2^{31} - 1$		
Max. precyzja				7 cyfr	14 cyfr

N słupków Od Do

```
root[1] TH1F h1 ("hist1", "My histogram", 100, -10., 10.);
root[2] h1.Fill (5.23);
root[3] h1.Draw ();
root[4] h1.Fill (3.21, 0.1); h1.Draw ();
root[5] h1.Draw ("hist");

root[6] TRandom3 r;  r.SetSeed ();
root[7] for (int i=0; i < 1e5 ; i++)  h1.Fill ( r.Gaus() );
root[8] h1.Draw ();
```

Wypełnienie słupka zawierającego x = 5.23, z wagą 1.

Wypełnienie słupka zawierającego x = 3.21, z wagą 0.1

Opcje rysowania

Dla TH1_ : `"same", "e", "e0 … 4", "hist", …`

Dla TH2_ : `"box", "col", "cont", "lego", "surf"` (i wersje, np. `"lego2"`)

→ root.cern/root/htmldoc/guides/users-guide/ROOTUsersGuide.html#draw-options

Opcje można łączyć, np. `h.Draw ("colz") , h.Draw ("e1same")`

Histogramy, c.d.

```
root[1] h1.Set<Tab>

root[1] h1.SetTitle ("Nowy tytul")
root[2] h1.SetMinimum (-20) ; h1.SetMaximum (200)

root[3] h1.SetLineColor (…) / SetLineStyle (…) / SetLineWidth (…)

root[4] h1.SetStats ( 0 / 1 );        <-- (Nie) wyświetlaj statystyk (szczegóły przy klasie TStyle)

root[5] h1.SetNdivisions ( kod , " Oś " );
```

Kod = `N1 + 100*N2 + 10000*N3`, gdzie:

N1: (oczekiwana) liczba podziałek głównych
N2: (oczekiwana) liczba podziałek 2. rzędu
N3: (oczekiwana) liczba podziałek 3. rzędu

```
root[6] h1.Get<Tab>

root[6] cout << h1.GetMean() << '\t' << h1.GetRMS() << '\t' << h1.GetNbinsX();
root[7] h1.Integral (<Tab>

Double_t Integral(Int_t binx1, Int_t binx2, Option_t* option = "") const
```

Napotykamy na problem. Jak znaleźć nr binu odpowiadający argumentowi? → metoda `FindBin(x)`

```
root[8] h1.Integral (h1.FindBin (-2.) , h1.FindBin(2.) )
```

Uwaga: Integral bez podania opcji – nie jest całkowaniem ($\Sigma h_i \cdot \Delta$) , tylko sumowaniem zawartości binów (Σh_i).
Aby wykonać całkowanie, podajemy opcję `"width"`.

Histogramy, c.d.

- Dostęp do zawartości histogramu.

Numeracja binów: [1 … h1.GetNbinsX()]

```
root[1] cout << h1.GetBinContent (41) <<'\t'<< h1.GetBinError (41) << endl;
11  3.31662
```

jak widać, niepewności są poissonowskie

```
root[2] Float_t* hy = h1.GetArray ()
```

Zwróci nam tablicę krotności.

Uwaga na numerację: hy[1] … hy[N]

W hy[0] / hy[N+1] tkwi tzw. *underflow / overflow*

- Niestety, nie ma metody odzyskującej tablicę niepewności...

```
root[3] Float_t* hyerr = new Float_t [h1.GetNbinsX() + 2]
root[4] for (int i=0; i<= h1.GetNbinsX()+1; i++) hyerr[i] = h1.GetBinError(i)
```

- Można też odzyskać położenia środków binów, jednak znów: brak jednej metody.

```
root[5] Float_t* hx = new Float_t [h1.GetNbinsX() + 2]
root[6] for (int i=0; i<= h1.GetNbinsX()+1; i++) hx[i] = h1.GetBinCenter (i)
```

- Tworzenie oddzielnej tablicy danych na niepewności (przed wypełnianiem; potrzebne przy manipulacjach) :

```
root[7] h1.Sumw2()
```

Histogramy, c.d.

Operacje na histogramach (Dodawanie, mnożenie, dzielenie)

```
root[1] TH1F::Add (
```

```
Bool_t Add(TF1* h1, Double_t c1 = 1, Option_t* option = "")
Bool_t Add(const TH1* h, const TH1* h2, Double_t c1 = 1, Double_t c2 = 1)
Bool_t Add(const TH1* h1, Double_t c1 = 1)
```

*this = c1*h + c2*h2*

*this += c1*h1*

```
root[2] TH1F h2 (h1); h2.Reset()
```

Nowy histogram: taki, jak h1, ale pusty.

```
root[3] for (int i=0; i < 1e5 ; i++)  h2.Fill ( r.Gaus (5., 1.) );
```

Wypełniamy r. Gaussa

```
root[4] h2.Draw();  h2.Add ( &h1 , 0.1 );  h2.Draw ("e1")
```

Dodajemy

```
root[5] TH1F h3 (h1); h3.Reset()

root[6] for (int i=1; i<1000; i++) h3.Fill (-9.99 + 0.02*i) ;
```

Rozkład jednorodny

```
root[7] h2.Multiply ( &h3 )
```

Mnożymy

```
root[8] h2.Draw ("e1")
```

Niepewności wg. prawa propagacji błędów

```
root[9] TH1F h4 (h1); h4.Reset ()
root[10] h4.Divide ( &h2 , &h3 )
```

Dzielimy

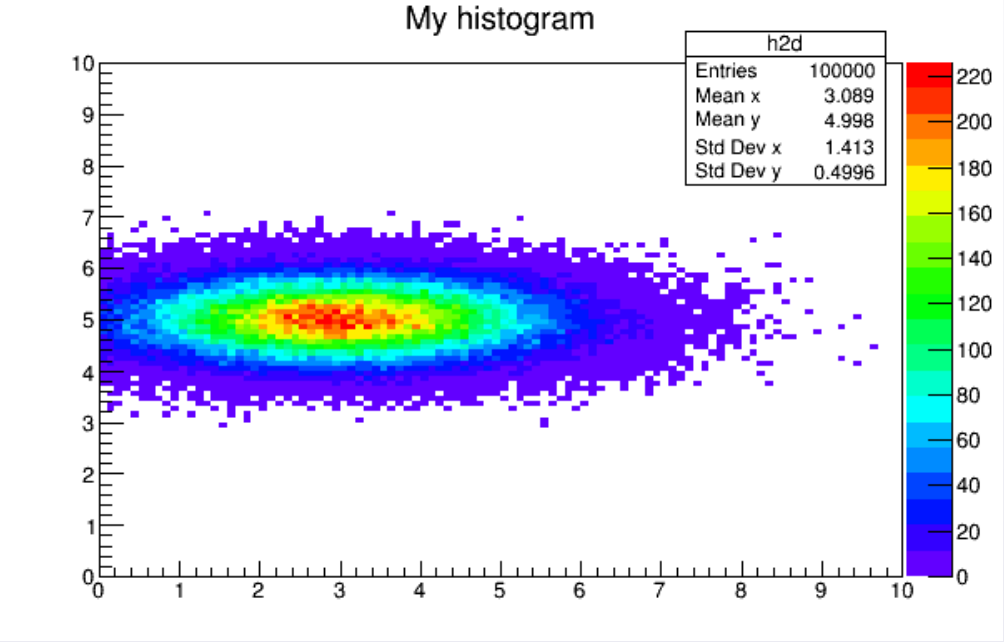
```
root[11] h4.Draw ("e1")
```

Niepewności narastają, nie niwelują się.

Histogramy 2-Dim

OŚ X			OŚ Y		
N słupków	Od	Do	N słupków	Od	Do

```
root[1] TH2F h2d ("h2d", "My histogram", 100, 0., 10., 100, 0., 10.);
root[2] for (int i=0; i<1e5; i++)  h2d.Fill (r.Gaus(3,1.5), r.Gaus(5,0.5));
root[3] h2d.Draw ("colz")
```




```
root[4] cout << h2d.GetNbinsX() << '\t' << h2d.GetNbinsY() << endl;
100     100
```

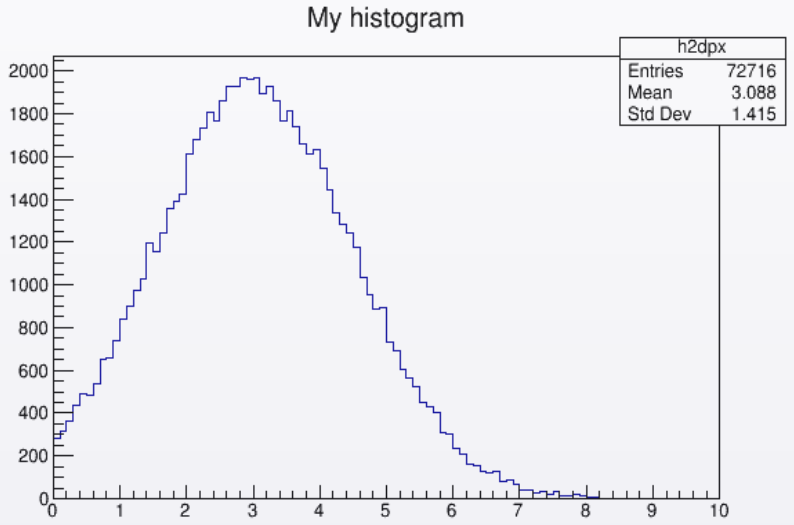
- Dostęp do brzegów każdej z osi (i innych danych o osi) – obiekt klasy **TAxis**

```
root[5] TAxis* ax = h2d.GetXaxis () , * ay = h2d.GetYaxis ();
root[6] cout << ax->GetXmin() <<' '<< ax->GetXmax() <<' '<< ax->GetNbins()<< endl;
root[7] cout << ay->GetXmin() <<' '<< ay->GetXmax() <<' '<< ay->GetNbins()<< endl;
root[8] cout << ax->GetBinWidth(1) <<' '<< ay->GetBinWidth(1) << endl;
```

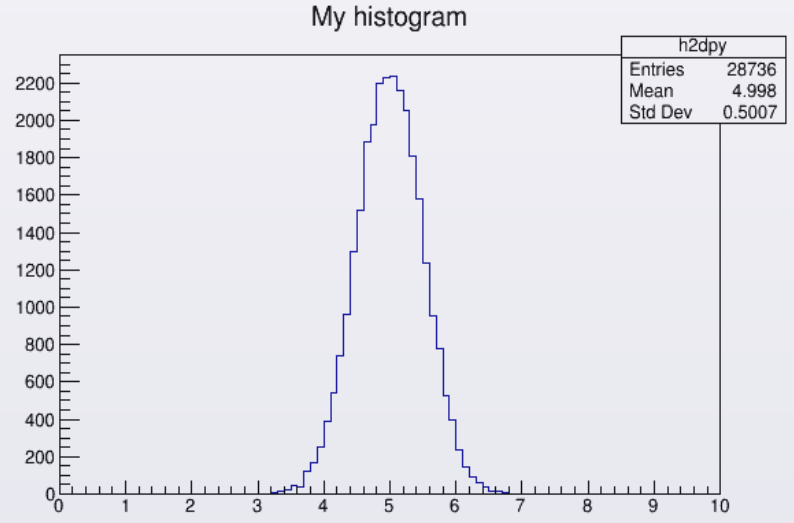
Histogramy c.d.

Projekcje 2dim → 1dim (na oś X lub oś Y)

```
root[8] h2d.ProjectionX ( "h2dpx", ay->FindBin(4.5) , ay->FindBin(5.5) )
h2dpx(class TH1D *) 0x3660560
root[9] h2dpx->Draw()
```




```
root[10] h2d.ProjectionY ( "h2dpy", ax->FindBin(2.5) , ax->FindBin(3.5) )
root[11] h2dpy->Draw()
```



Podstawowa grafika

- Punkty/Markery (`TMarker`), Linie (`TLine`) , Strzałki (`TArrow`)
- Prostokąty (`TBox`) , Okręgi/elipsy (`TEllipse`)
- Napisy (`TText`) , w stylu LaTeX (`TLatex`)

Przykładowa pomoc

→ `Manual:` `rozdział Graphics`
→ `User guide:` `rozdział Graphics`

Podstawowe obiekty

[1] **TLine** (Linie)

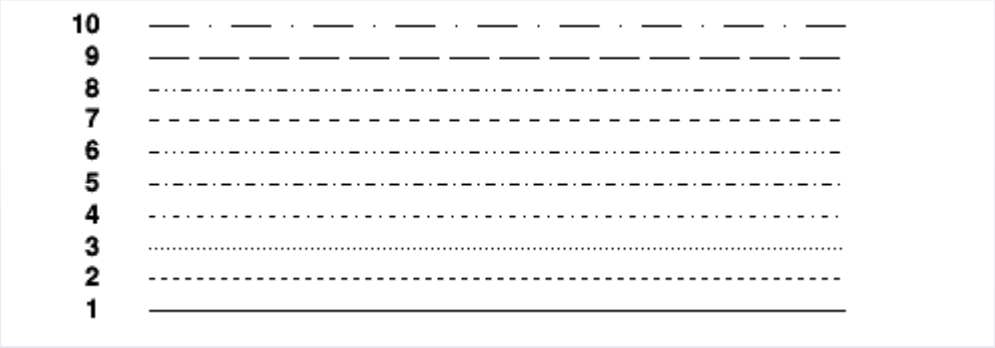
```
root[1] TH2F h2d ("h2d", "My Histo", 10, 0., 10., 10, 0., 10.); h2d.Draw()
root[2] TLine l1 (0., 0., 1., 1.) ; l1.Draw ()
root[3] TLine l2 (0., 0., 1., 1.) ; l2.SetNDC (kTRUE) ; l2.Draw("same")
```

NDC (Normalized Device Coordinates) :

```
SetNDC (0)    ← Współrzędne aktualnego wykresu
SetNDC (1)    ← Współrzędne okna graficznego
```

```
root[4] l2.SetLineColor (2); l2.SetLineStyle (2); l2.SetLineWidth (3)
```

Numeracja kolorów w palecie generycznej



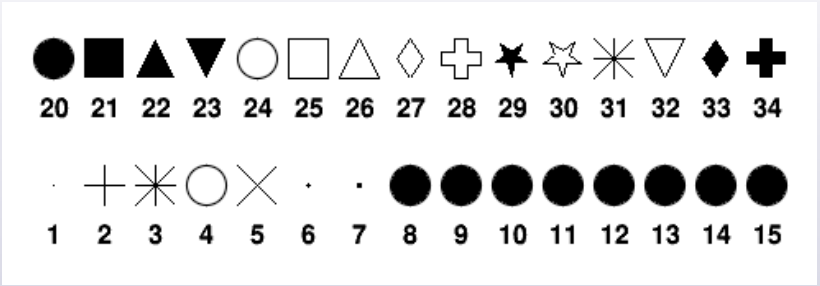
Numeracja stylów linii

Podstawowa grafika c.d.

Podstawowe obiekty

[2] TMarker (Punkty / markery)

```
root[1] TMarker m (3., 8., 31)
root[2] m.SetMarkerColor (4);
root[3] m.SetMarkerSize (2.0);
root[4] m.Draw()
```



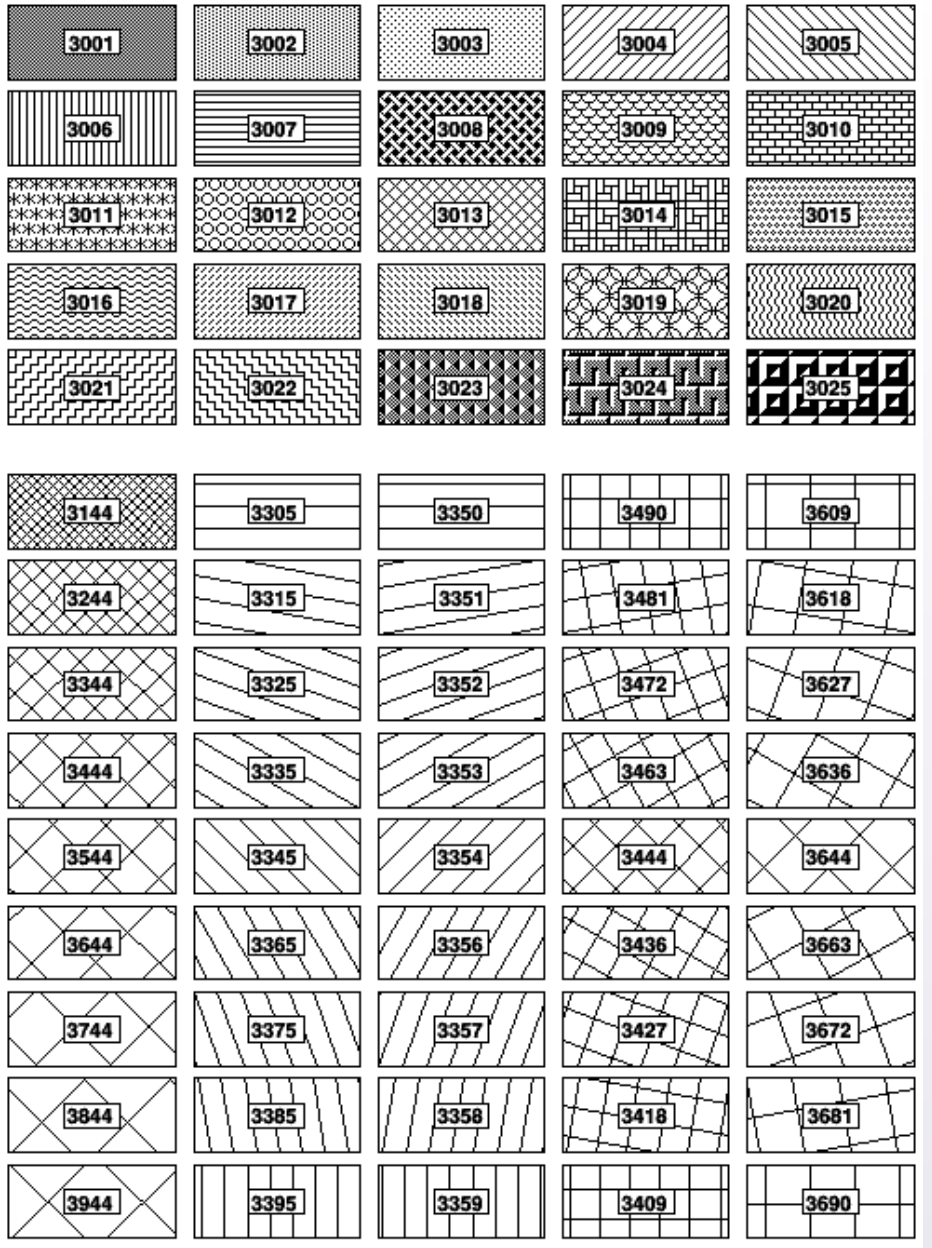
Numeracja stylów markerów

[3] TBox (Prostokąty)

```
root[1] TBox b (5., 2., 8., 3.);
root[2] b.SetLine.....
root[3] b.SetFillColor (4)
root[4] b.SetFillStyle (3014);
```

Konwencje określania stylu wypełnienia:

root.cern/doc/master/classTAttFill.html#ATTFILL2



Numeracja niektórych stylów kreskowania

Podstawowa grafika c.d.

[4] TEllipse (Okręgi / elipsy)

```
root[1] TEllipse e (
TEllipse TEllipse(Double_t x1, Double_t y1,
Double_t r1, Double_t r2 = 0,
Double_t phimin = 0, Double_t phimax = 360,
Double_t theta = 0)
```

Zakres katów wycinka elipsy: [PhiMin, PhiMax]
Kąt obrotu figury: Theta

```
root[1] TEllipse e (5, 5, 4, 2, 0, 270, 45)
```

Działają atrybuty linii i wypełnienia

[5] TText (Tekst podstawowy)

```
root[1] TText t (0.5, 0.5, "Hello World !");
root[2] t.SetTextColor (2)
root[3] t.SetTextFont (43)
root[4] t.SetTextSize (40)
root[5] t.SetTextAngle(45)
root[6] t.Draw()
```

Numeracja stylów czcionek.
Cyfra jednostek określa stopień precyzji.

12 : *ABCDEFGH abcdefgh 0123456789 @#$*

22 : **ABCDEFGH abcdefgh 0123456789 @#$**

32 : ***ABCDEFGH abcdefgh 0123456789 @#$***

42 : ABCDEFGH abcdefgh 0123456789 @#$

52 : *ABCDEFGH abcdefgh 0123456789 @#$*

62 : **ABCDEFGH abcdefgh 0123456789 @#$**

72 : ***ABCDEFGH abcdefgh 0123456789 @#$***

82 : ABCDEFGH abcdefgh 0123456789 @#$

92 : *ABCDEFGH abcdefgh 0123456789 @#$*

102 : **ABCDEFGH abcdefgh 0123456789 @#$**

112 : ***ABCDEFGH abcdefgh 0123456789 @#$***

122 : ΑΒΧΔΕΦΓΗ αβχδεφγη 0123456789 ≅#∃

132 : ABCDEFGH abcdefgh 0123456789 @#$

142 : ♏♐♑♒ 🗀🗁🗎🗐 ✍✂

152 : *ΑΒΧΔΕΦΓΗ αβχδεφγη 0123456789 ≅#∃*

Podstawowa grafika c.d.

[6] **TLatex** (napisy zaawansowane: wyrażenia matematyczne itp)

Przydatne streszczenie: root.cern.ch/doc/master/classTLatex.html

```
root[1] TLatex l;
root[2] l.SetNDC (1);
root[3] l.SetTextSize (0.06);
root[4] l.SetTextAngle (45.);
root[5] l.SetTextColor (4);
root[6] l.DrawLatex (0.5, 0.6, "E^{2} = m^{2} + p^{2}")
```

← Jeden obiekt zawiaduje serią napisów

Kilka prostych zasad:

`^{…}`	: indeks górny	`#frac{Licznik}{Mianownik}`	: ułamek poziomy
`_{…}`	: indeks dolny	`#sqrt{x}  ,  #sqrt{N}{x}`	: pierwiastek stopnia 2 i wyżej
`#bf{…}`	: pogrubienie	`#splitline{Gora}{Dol}`	: dwie linie
`#it{…}`	: kursywa	`#color[4]{Niebieski}`	: lokalna zmiana koloru
`#vec{…}`	: wektor	`#font[12]{Czcionka}`	: lokalna zmiana kroju czcionki
`#(){…}`	: duże nawiasy	`#scale[1.2]{Wiekszy}`	: lokalne skalowanie czcionki

```
root[7] l.DrawLatex (0.5,0.6,"#gamma_{cm} = #frac{1}{#sqrt{1-#beta^{2}_{cm}}}")
```

Przykłady ze strony źródłowej:

{}^{40}_{20}Ca : $^{40}_{20}Ca$

x = #frac{y+z/2}{y^{2}+1} : $x = \frac{y+z/2}{y^2+1}$

Podstawowa grafika c.d.

[6] Klasa **TLatex** c.d.

Kody symboli greckich (poprzedzamy przez #)

Lower case		Upper case		Variations	
alpha :	α	Alpha :	Α		
beta :	β	Beta :	Β		
gamma :	γ	Gamma :	Γ		
delta :	δ	Delta :	Δ		
epsilon :	ϵ	Epsilon :	Ε	varepsilon :	ε
zeta :	ζ	Zeta :	Ζ		
eta :	η	Eta :	Η		
theta :	θ	Theta :	Θ	vartheta :	ϑ
iota :	ι	Iota :	Ι		
kappa :	κ	Kappa :	Κ		
lambda :	λ	Lambda :	Λ		
mu :	μ	Mu :	Μ		
nu :	ν	Nu :	Ν		
xi :	ξ	Xi :	Ξ		
omicron :	ο	Omicron :	Ο		
pi :	π	Pi :	Π		
rho :	ρ	Rho :	Ρ		
sigma :	σ	Sigma :	Σ	varsigma :	ς
tau :	τ	Tau :	Τ		
upsilon :	υ	Upsilon :	Υ	varUpsilon :	ϒ
phi :	ϕ	Phi :	Φ	varphi :	φ
chi :	χ	Chi :	Χ		
psi :	ψ	Psi :	Ψ		
omega :	ω	Omega :	Ω	varomega :	ϖ

Górne znaki diakrytyczne

#tilde :	ã
#ddot :	ä
#dot :	ȧ
#grave :	à
#acute :	á
#check :	ǎ
#hat :	â
#bar{a} :	$\bar{a}$
#vec{a} :	$\vec{a}$

Podstawowa grafika c.d.

[6] Klasa **TLatex** c.d.

Symbole matematyczne i inne symbole specjalne

♣	#club	⇔	#Leftrightarrow	←	#leftarrow	∇	#nabla	å	#aa
℘	#voidn	|	#void8	⊗	#otimes	↵	#downleftarrow	⁄	#/
≤	#leq	ℏ	#hbar	⇐	#Leftarrow	—	#topbar	\	#backslash
≈	#approx	♦	#diamond	∏	#prod	⌠	#arcbar	·	#upoint
∈	#in	ℵ	#aleph	□	#Box	↑	#uparrow	∂	#partial
⊃	#supset	≥	#geq	||	#parallel	⊕	#oplus	¬	#corner
∩	#cap	≠	#neq	♥	#heart	⇑	#Uparrow	⌇	#ltbar
©	#ocopyright	∉	#notin	ℑ	#Jgothic	∑	#sum	⌊	#bottombar
™	#trademark	⊆	#subseteq	⟨	#LT	⊥	#perp	→	#rightarrow
×	#times	∪	#cup	≡	#equiv	∀	#forall	√	#surd
•	#bullet	℗	#copyright	⊂	#subset	♠	#spade	⇒	#Rightarrow
ƒ	#voidb	™	#void3	⊇	#supseteq	ℜ	#Rgothic	∫	#int
″	#doublequote	÷	#divide	∧	#wedge	⟩	#GT	⊙	#odot
|	#lbar	°	#circ	®	#oright	∝	#propto	∃	#exists
⌈	#arcbottom	∞	#infty	Å	#AA	⊄	#notsubset	+	#plus
↓	#downarrow	∠	#angle	±	#pm	∅	#oslash	−	#minus
↔	#leftrightarrow	|	#cbar	∓	#mp	∨	#vee		
⇓	#Downarrow	⌠	#arctop	⋯	#3dots	®	#void1		

TStyle Obiekt stylu graficznego

- W sesji ROOT'a dostępny jest obiekt **gStyle** klasy **TStyle** z ustawieniami stylu grafiki. Ustawienia dotyczą m.in. atrybutów obrazka, linii, markerów, wypełnień, statystyk. Dostęp jest przez getter'y i setter'y.

```
root[1] gStyle->Set
```


```
root[1] gStyle->SetLabelSize   (0.07, "XY");
root[2] gStyle->SetLabelOffset (0.01, "Y" );
root[3] gStyle->SetNdivisions  ( 2  , "X" );
root[4] TH1F h1 ("h1", "", 10, -5, 5);
root[5] h1.Draw ();
```

- Jeśli jednak najpierw zdefiniowaliśmy histogram, a później zmieniliśmy styl, to musimy powiadomić histogram, aby ten styl zaktualizował:

```
root[6] gStyle->SetNdivisions ( 8 , "X" );
root[8] h1.Draw ();
root[9] h1.UseCurrentStyle ();
root[10] h1.Draw ();
```

- Ważne: `TStyle` umożliwia modyfikację wyświetlanych statystyk histogramu:

```
root[11] gStyle->SetOptStat ( {rmen} );
```

Symbole {r m e n} to podstawowe 4 z 9 atrybutów do wyświetlenia. Przyjmują wartości {0, 1}, czasem 2.

Podstawowa lista:

```
r = (0) 1:   (nie) wyświetla RMS
m = (0) 1:   (nie) wyświetla średnią
e = (0) 1:   (nie) wyświetla liczbę przypadków
n = (0) 1:   (nie) wyświetla identyfikator histogramu
```

Link do pełnej listy.

TFile Pliki .root do zapisu obiektów

(root.cern/root/htmldoc/guides/users-guide/InputOutput.html
root.cern/doc/master/classTFile.html)

Na start:

```
root[1] TFile* f = new TFile ("mojplik.root", "RECREATE");
root[2] TH1F* h = new TH1F ("myhisto", "My Histo's Title", 10, -5., 5.);
root[3] TRandom3 r;  for (int i=0; i<1e5; i++) h->Fill ( r.Gaus() );
root[4] h->Write ();                                          <-- Wpisujemy obiekt do pliku
root[5] f->Close ();
root[6] .!ls -l mojplik.root

root[1] f = new TFile ("mojplik.root", "READ");               <-- Będziemy czytać z pliku
root[2] TH1F* hread = (TH1F*) f-> Get ("myhisto");
root[3] hread->Draw ();
root[4] f->Close ();
```

Bliższe spojrzenie:

```
root[1] TFile* f = new TFile ("mojplik.root", option);
                                   option = "NEW" "RECREATE" "UPDATE" "READ"
root[2] if (f->IsOpen() == true) cout << "File open.\n";
root[2] f->ls();
root[3] TH1F* h = new TH1F ("h", "myhisto", 10, 0., 10.);
root[4] f->ls();                                              <-- h podpięty, ale niezapisany.
root[5] .!ls -l mojplik.root
root[6] h->Write ();                                          <-- h podpięty i zapisany.
root[7] f->ls();
root[8] .!ls -l mojplik.root
root[9] h->Write ("h_kopia");                                 <-- h pod nowym identyfikatorem
root[10] f->ls ();
root[11] f->Close();
```

I/O c.d.

Sesja ROOT'a z podpięciem pliku:

```
> nice root -l mojplik.root
root[0]
Attaching file mojplik.root as _file0...
(class TFile *) 0x1943c70
root[1] _file0->ls();
```

← Wskaźnik do obiektu klasy `TFile`

Ustawienie ścieżki pracy na dysku:

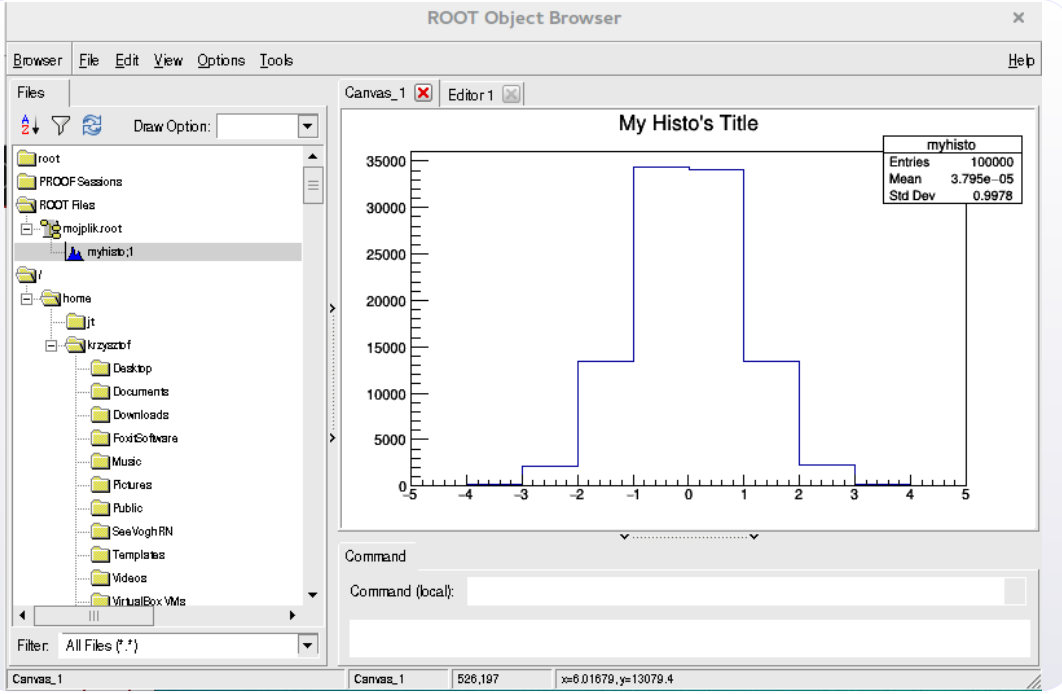
```
root[2] gSystem->pwd ()
(const char *) "/home/krzysztof/didact/informatyka/nuctools"
root[3] gSystem->cd ("../")
root[4] gSystem->pwd()
(const char *) "/home/krzysztof/didact/informatyka/"
```

Wykonanie komendy w Linux'ie :

```
root[4] gSystem->Exec ("date")
Wto, 5 lis, 09:55:01 CET

root[5] TString datenow =
        gSystem->GetFromPipe ("date")
```

Przeglądarka obiektów TBrowser

```
root[5] new TBrowser
```



gDirectory Wewnętrzna struktura obiektów

ROOT posiada **strukturę katalogów** (w pamięci oraz w plikach .root) .

```
> nice root -l
root[0] gDirectory->pwd ()      lub      .pwd
Current directory: Rint:/
```

← Główny katalog w pamięci

```
root[1] TFile f1 ("mojplik.root"); cout << gDirectory->GetPath () << endl;
Mojplik.root:/
root[2] gDirectory->ls ()       lub      .ls
TFile**         mojplik.root
 TFile*         mojplik.root
  KEY: TH1F     h;1     myhisto
```

← Przeszliśmy do pliku

Tworzenie podścieżek (w pamięci lub w pliku .root)

```
root[3] TFile f2 ("newfile.root", "RECREATE")
root[4] gDirectory->mkdir ("katalog1");
root[5] gDirectory->cd ("katalog1");  gDirectory->pwd ()
newfile.root:/katalog1

root[6] TH1F h ("myhisto", "", 10, -5., 5);  h.Write();
root[7] .ls
TDirectoryFile*         katalog1     katalog1
 OBJ: TH1F      myhisto  : 0 at: 0x7f06ee3ce000
 KEY: TH1F      myhisto;1
```

← Nowy katalog w pliku

← myhisto zapisane w katalog1

```
root[8] gDirectory->cd ("..") ;  gDirectory->rmdir ("katalog1")
root[8] f2.Close ();
root[9] cout << gDirectory->GetPath() << endl;
Rint:/
Root[10] gDirectory->cd ("Rint:/");  gDirectory->pwd();
```

← Powrót do kat. głównego

MAKRA : Kody C++/ROOT w pliku

Mod podręczny: `macro_noname.C`

```
{
  TH1F h1 ("hist1", "", 50, -5., 5.);
  TH1F* h2 = new TH1F
           ("hist2", "", 50, -5., 5.);

  TRandom3 r;  r.SetSeed ();

  for (int i=0 ; i<1e5 ; i++) {
    h1.Fill  ( r.Gaus() );
    h2->Fill ( r.Gaus() );
  }
  h1.Draw();
}
```

Po wykonaniu, w sesji interaktywnej:

- Funkcjonuje: obiekt `h1` i wskaźnik `*h2`
- Można się odwołać: `h1, h2, hist1, hist2`

Mod pełny (funkcje) — Nazwa jak nazwa pliku

```
int macro_function () {
  TH1F h1 ("hist1", "", 50, -5., 5.);
  TH1F* h2 = new TH1F
           ("hist2", "", 50, -5., 5.);

  TRandom3 r;  r.SetSeed ();

  for (int i=0 ; i<1e5 ; i++) {
    h1.Fill  ( r.Gaus() );
    h2->Fill ( r.Gaus() );
  }

//TCanvas can1 ("c1", "", 640, 480);

  h1.Draw();

//can1.Update();
//cin.ignore();

  return 0;
}
```

Po wykonaniu, w sesji interaktywnej:

- Nie ma obiektu `h1` (również przez `hist1`)
- Nie można wywołać `h2`
- MOŻNA wywołać `hist2`

Dodatkowe komendy do przechwycenia grafiki

MAKRA c.d.

Argumenty wejścia makra:

```
double macro_inputarg (double x = 0)
{
  cout << "Hello world! " << x << endl;
  return x;
}
```

```
> nice root -b "macro_inputarg.C(12.34)"
```

← Bez spacji między .C a (...)

Wywołanie z poziomu sesji:

```
root[0] .x macro_inputarg.C (-12.)
```

Makro może mieć wiele funkcji.
Funkcja startowa MUSI mieć nazwę tożsamą z nazwą pliku (odpowiednik `main`)

```
Double_t W (Double_t x) {
  return 3*pow(x,2) - 1.5*x + 4.;
}

int macro_giveW (double x) {
  cout << "W(x) = " << W(x) << endl;
  return 0;
}
```

Makro można najpierw załadować, a wykonać później.

```
root[0] .L macro_inputarg.C
root[1] macro_inputarg (123) ;
Hello world! 123
```

W sesji można makro skompilować. Jednak kod musi zawierać `#include< … >`

```
root[0] .L macro_inputarg.C+
root[1] macro_inputarg (-123.45);
```

← Utworzy macro_inputarg_C.so

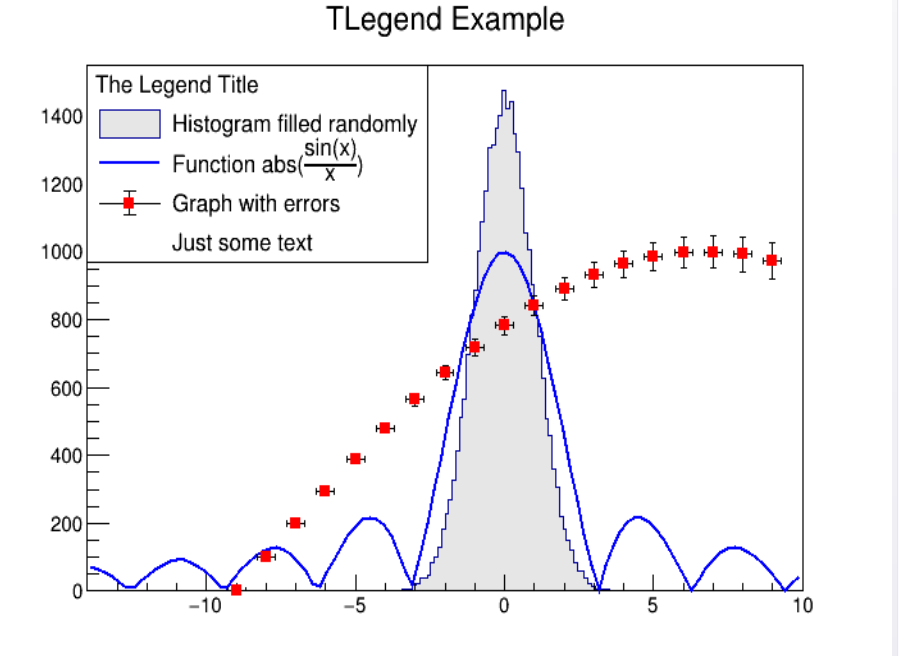
TLegend Legenda wykresu

Do każdego wykresu można dodać legendę (jedną lub więcej).
Obiekt klasy **TLegend** łączy się z narysowanymi na wykresie: funkcjami, histogramami lub grafami.
My decydujemy, jak je podpisać i zasymbolizować (linia, marker, prostokąt, punkt z niepewnością).

```
int macro_TLegend () {
  TH1F* h = new TH1F ("h", "Example", 200, -14, 10);
  h->FillRandom ("gaus", 30000);
  h->SetFillColor (18);
  h->Draw ();

  TF1* f= new TF1("f", "1000*abs(sin(x)/x)",-14,14);
  f->SetLineColor (kBlue);
  f->Draw ("same");

  Int_t i = 0;
  Double_t x[50], y[50], ex[50], ey[50];
  for (Double_t xval = -9; xval <= 9; xval++, i++){
    x[i]  = xval;
    y[i]  = 1000 * sin ( (xval + 9)/10 ) ;
    ex[i] = 0.3;
    ey[i] = (xval+9) * 3 ;
  }
  TGraphErrors* gr = new TGraphErrors (i,x,y,ex,ey);
  gr->SetName ("gr");
  gr->SetMarkerStyle (21);
  gr->SetMarkerColor (kRed);
  gr->Draw ("P");

  TLegend* leg = new TLegend (0.1, 0.6, 0.48, 0.9);
  leg->SetHeader ("The Legend Title");
  leg->AddEntry ( h  ,"Histogram filled rand", "f");
  leg->AddEntry ("f" ,"abs(#frac{sin(x)}{x})", "l");
  leg->AddEntry ("gr","Graph with errors"  , "lep");
  leg->AddEntry ((TObject*)0, "Just some text", "");
  leg->Draw ();
  return 0;
}
```

Legendę trzeba wyświetlić poleceniem.

To makro rysuje histogram, funkcję i `TGraph`.
Następnie tworzy do nich legendę
i modyfikuje atrybuty symboli w legendzie.



Utworzenie legendy w danych wymiarach
Tytuł legendy

Wpisy dla obiektów.
Opcje: f = filled box, l = line
p = marker e = error bar

Gdy chcemy tylko tekst

Dobry wykres w ROOT

Niniejsze makro ukazuje propozycję drogi utworzenia czytelnego, przejrzystego wykresu.

```
int macro_GoodPlotExample () {
  gStyle->SetOptStat (0);
  gStyle->SetLegendBorderSize (0);
  gStyle->SetLegendTextSize   (0.055);
  gStyle->SetLabelSize  (0.055, "XY");
  gStyle->SetNdivisions ( 505 , "XY");

  {… creation of histogram, curve and TGraphErrors …}

  TCanvas* c1 = new TCanvas ("c1", "", 800, 600);
  c1->SetGrid (0, 0);
  c1->SetTopMargin    (0.05);
  c1->SetBottomMargin (0.15);
  c1->SetRightMargin  (0.04);
  c1->SetLeftMargin   (0.16);

  h1->Draw ();
  f1->Draw ("same");
  gr->Draw ("P");

  TLegend* leg = new TLegend
           (0.21, 0.63, 0.5, 0.86, "", "nbNDC");
  leg->AddEntry ( h  , "   Experiment" , "f"  );
  leg->AddEntry ("f" , "   Model"      , "l"  );
  leg->AddEntry ("gr", "   Efficiency" , "lep");
  leg->Draw ();

  TLatex l;
  l.SetNDC (1);
  l.SetTextSize (0.055);
  l.DrawLatex (0.44, 0.025, "Position  (#mum)");
  l.SetTextAngle (90);
  l.DrawLatex (0.035, 0.34 , "Rate  (arb. units)");
  l.SetTextAngle (0);
  return 0;
}
```

Ustawienia stylu.

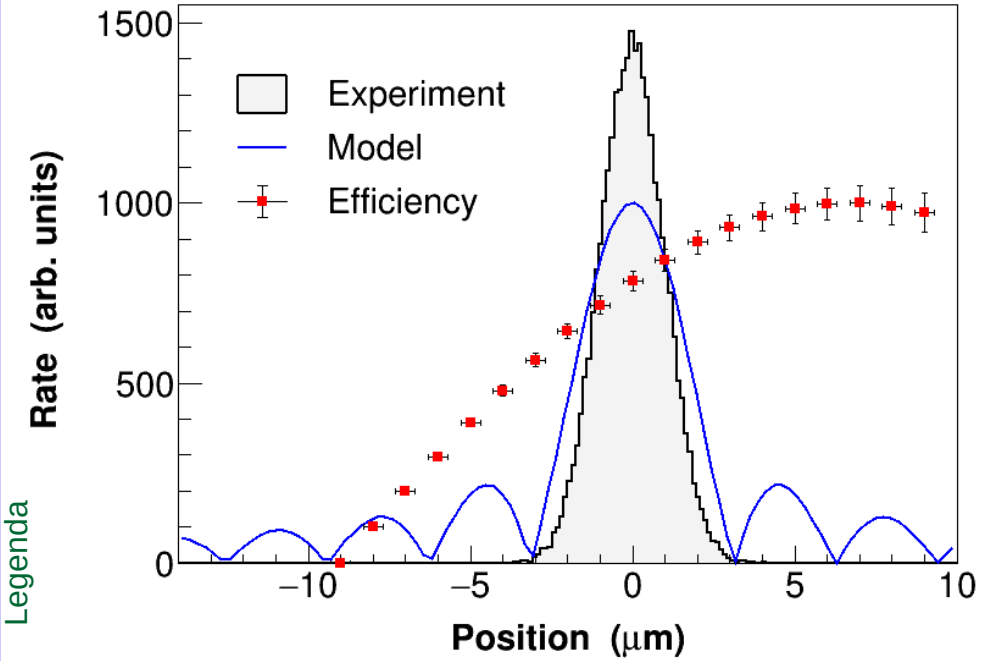
Nie wyświetlaj statystyk.
Legenda: bez ramki.
Legenda: rozmiar czcionki.
Rozmiar czcionki wartości przy osiach
Ilość podziałek na osiach

Marginesy

Legenda

Podpisy osi

Używaj współrzędnych obrazka
Rozmiar czcionki
Nadrukuj podpis pod osią poziomą
Zmiana kąta na podpis pionowy
Nadrukuj podpis pod osią pionową



MAKRA : Funkcje C++ używane do funkcji TFn

Poprzez "Inline Expression" :

```
Double_t W (Double_t x) {
  return pow(x,3) - 6.*pow(x,2)
         + x - 1.;
}

int macro_funcInline ()
{
  TCanvas c1 ("c1", "", 640, 480);
  TF1 f1 ("f1", "W(x)", -10., 10.);
  f1.Draw();
  c1.Update();
  cin.ignore();
  return 0;
}
```

Poprzez funkcję z parametrami:

```
Double_t myFun (Double_t *xarg,
                Double_t *par)
{
  Double_t x = xarg[0] , result = 0.;

  for (int st=0; st<=3; st++)
    result += par[st]
                * TMath::Power (x, st);
  return result;
}

int macro_funcFunc ()
{
  TCanvas c1 ("c1", "", 640, 480);
  TF1 f1 ("myfun1", myFun, -3, 5, 4);
  f1.SetParameters (-1., 1., -6., 1.);
  f1.Draw();
  c1.Update();
  cin.ignore();
  return 0;
}
```

Uwaga: w `myFun` możemy zakodować wszystko,
m.in. instrukcje warunkowe
i wywołania innych funkcji

Dopasowywanie funkcji do punktów pomiarowych

Przykładowy zestaw danych: `wget www.fuw.edu.pl/~kpias/ctnp/dataPoints.txt`

```
void macro_FitTGraphErrors () {
 TCanvas* c1 = new TCanvas ("c1");
 TGraphErrors gr ("dataPoints.txt");
 gr.SetTitle ();
 gr.Draw("AP");

 TF1 fun ("fun", myFun , -3, 5, 4);
 fun.SetParameters (-1. , 1. , -6. , 1.);
 gr.Fit ( &fun );

 c1->Update();
 cin.ignore();
}
```

Graf z danymi doświadczalnymi

Funkcja modelowa (z parametrami - !)

Dopasowanie. Pakiet MINUIT (CERN)

Przykładowy wynik (skutecznego) dopasowania:

χ^2

Stan fitu (✓)

```
FCN=10.8023 FROM MIGRAD    STATUS=CONVERGED      91 CALLS          92 TOTAL
                     EDM=5.47088e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  p0           -1.31434e+00   3.32403e-01   3.54860e-04   1.79927e-03
  2  p1            1.25659e+00   3.08140e-01   2.52787e-04   4.11222e-03
  3  p2           -5.75515e+00   1.76173e-01   1.13715e-04   1.26442e-02
  4  p3            9.12643e-01   7.84954e-02   4.27100e-05   3.82217e-02
```

Status macierzy kowariancji (✓)

wyniki dla parametrów

Wartości najlep. dopasowania

Niepewności

Dopasowywanie funkcji do punktów pomiarowych c.d.

Ustawianie parametrów startowych:

```
fun1.SetParameter (numer, wartość) ;                    ← Ustaw 1 parametr
fun1.SetParameters (wartość, wartość, … , wartość);     ← Ustaw parametry
fun1.FixParameter (numer, wartość) ;                    ← Zablokuj 1 parametr

fun1.SetParLimits (numer, min, max) ;                   ← Ustaw zakres dopasowania
gr.Fit ( &fun , "" );
```

Pobieranie wyników dopasowania:

```
fun1.GetParameter (numer) ;      ← Pobierz 1 parametr
fun1.GetParError (numer) ;       ← Pobierz niepewność
fun1.GetChisquare ();            ← Pobierz χ²
fun1.GetNDF ();                  ← Pobierz l. stopni swobody
```

Użycie funkcji wbudowanych (predefiniowanych) :

```
gr.Fit ( "pol3" );
```

polN	:	f(x) = p0 + p1*x + p2*x2 + ...
gaus	:	f(x) = p0*exp (-0.5*((x-p1)/p2)^2)
gausn	:	(Rozkład Normalny)
expo	:	f(x) = exp (p0+p1*x)
landau	:	(rozkład Landaua strat energii)
chebyshevN	:	(wielomian Czebyszewa stopnia N)

Uwaga: dla funkcji *predefiniowanych*, jeśli zawężamy zakres parametrów (lub blokujemy wartość), w opcji metody `Fit` musimy wpisać `"B"` .

Dopasowywanie funkcji do punktów pomiarowych c.d.

Metoda **`Fit`** działa również dla histogramów, w tym 2, 3 – wymiarowych. Pełna postać metody:

```
TFitResultPtr Fit(TF1* f1, Option_t* option = "",          ← Opcje dopasowania
                           Option_t* goption = "",         ← Opcje rysowania
                  Double_t xmin = 0, Double_t xmax = 0)    ← Zakres na osi X
```

Opcje dopasowywania (wybór praktyczniejszych; szczegóły *w linku*)

wyłącznie dla histogramów (`THdf`) :

`I`	`(Integral)`	Uśrednij funkcję w binie (gdy funkcja się wyraźnie zmienia wzdłuż binu)
`L`	`(LogLikelihood)`	Użyj metody największej wiarygodności, Log Likelihood (zamiast χ^2).

Dla histogramów i grafów (`TGraph_____`) :

`M`	`(iMprove)`	Wyznacz dokładniejsze wyniki dopasowania
`E`	`(Error)`	Wyznacz dokładniejsze niepewności metodą MINOS pakietu Minuit.
`B`	`(Bound)`	Dla f. predefiniowanych: gdy zakres zmienności parametrów jest ograniczony
`R`	`(Range)`	Dopasuj w zakresie, w jakim zdefiniowana jest funkcja
`0`		Nie rysuj dopasowania na wykresie
`V`	`(Verbose)`	Tryb informatywny (maksimum monitów)
`Q`	`(Quiet)`	Tryb cichy (minimum monitów)

W przypadku dopasowywania histogramu 2 (3) – wymiarowego do funkcji 2 (3) wymiarowej:

- Przedział dziedziny, w której dopasowujemy, trzeba określić przy deklaracji TFn, w konstruktorze
- W opcji metody `Fit` dodać `"R"`.
- Ewentualne podanie `xmin` i/lub `xmax` w argumentach metody `Fit` działa tylko dla osi X

Dopasowywanie funkcji do punktów pomiarowych c.d.

- Pobranie **macierzy kowariancji** (*cel: korelacje liniowe między dopasowywanymi parametrami*)

```
void macro_FitErrorMatrix () {
 TCanvas* c1 = new TCanvas ("c1");
 TGraphErrors gr ("dataPoints.txt");
 gr.SetTitle ();
 gr.Draw("AP");

 TF1 fun ("fun", myFun , -3, 5, 4);
 fun.SetParameters (-1. , 1. , -6. , 1.);

 TFitResultPtr fitRes = gr.Fit ( &fun , "S" );
 TMatrixDSym cov = fitRes->GetCovarianceMatrix();

 for (int r = 0; r < cov.GetNrows () ; r++) {
  for (int c = 0; c < cov.GetNcols() ; c++)
   cout << setw(16) << cov[r][c] ;

  cout << endl;
 }

 c1->Update();
 cin.ignore();
}
```

`TFitResultPtr` przechowuje wyniki dopasowania

`TMatrixDsym` macierz symetryczna elementów *double*

`GetCovariance...` pobranie macierzy kowariancji

- Dostęp w kodzie do **statusu dopasowania** (*string*)

```
#include "TMinuit.h"
string myFitStatus = gMinuit->fCstatu ;
```

Na początku kodu

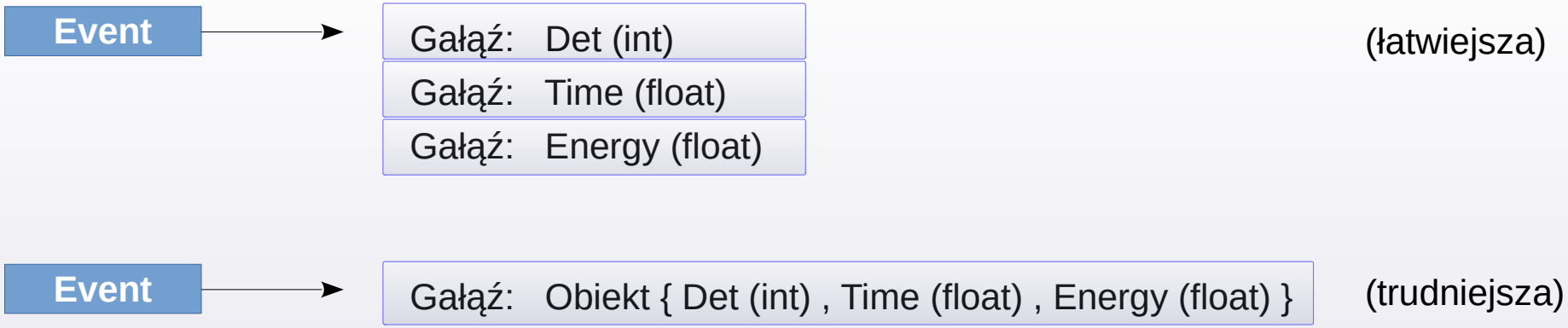
Wydobycie statusu (string)

TTree Drzewa (bazy danych)

Np. eksperyment rejestrujący cząstki w detektorach: zbiór N_i , T_i , ΔE_i , E_i z detektora (detektorów)
Np. eksperyment rejestrujący tory cząstek w komorze dryfowej : zbiór p_x, p_y, p_z, ΔE_i z detektora

Prosty schemat danych

Możliwe struktury eventu:



Zajmiemy się najpierw wersją 1.

TTree Jak najprościej stworzyć drzewo z danych tekstowych

① Ściągnij dane. Są one w 3 kolumnach (int, float, float)

```
wget www.fuw.edu.pl/~kpias/ctnp/MyExpData.txt
```

```
11      16.832    10.8703
11     20.4335    8.65938
 7    0.634218    8.03354
 1     21.3472    19.6014
...
```

② Otworzymy obiekt `TTree` i metodą `ReadFile` wypełnimy bazę:

Deklarujemy drzewo

`ReadFile` wypełnia drzewo z danych txt

`Print` wypisuje statystyki

`Scan` wypisuje dane

`Draw` : wykres `en-time` nakładamy filtr na `det`

`Write` zapisuje drzewo do pliku

```
int TTree_ReadFile ()
{
  TFile f ( "simplest_tree.root", "RECREATE" );
  TTree t ( "mytree" , "Tree of data for my analysis" );
  t.ReadFile ("MyExpData.txt", "det/I:energy/F:time/F" );
  t.Print ();
  t.Scan ( "det:energy:time" );
  t.Draw ( "energy:time" , "det >= 7" );
  t.Write ();
  return 0;
}
```

TTree c.d.

Teraz zaprojektujemy drzewo sami.

- **Schemat "1 gałąź = 1 zmienna"**

Definiujemy gałąź:

```
t.Branch ("Nazwa"   ,
          &zmienna,
          "zmienna/F");
```

Kody rozmiaru zmiennej:

F	: float	, 4 bajty
D	: double	, 8 bajtów
I	: signed integer	, 4 bajty
i	: unsigned integer,	4 bajty
C	: c-string	
B	: signed integer	, 1 bajt
b	: unsigned integer,	1 bajt
S	: signed integer	, 2 bajty
s	: unsigned integer,	2 bajty
L	: signed integer	, 8 bajtów
l	: unsigned integer,	8 bajtów
O	: bool	, 1 bit

```
int TTree_simple () {
  Int_t det;
  Float_t energy , time;

  TFile f ("simple.root", "RECREATE");
  TTree t ("tree", "My tree");
  t.Branch ("Det" , &det   , "det/I");
  t.Branch ("En"  , &energy, "energy/F");
  t.Branch ("Time", &time  , "time/F");

  TRandom3 r; r.SetSeed ();
  for (int i = 0; i < 100; i++) {
    det    = r.Integer (24);
    time   = r.Rndm() * 20.;
    energy = r.Rndm() * 30.;
    t.Fill ();
  }
  t.Write ();
  return 0;
}
```

Zapisujemy event do drzewa: `t.Fill ()`

Zapisujemy drzewo do pliku: `t.Write ()`

TTree c.d.

Podgląd drzewa w konsoli

```
nice root -l simple.root
root[0] tree->Print ()
******************************************************************************
*Tree    :tree      : My tree                                                  *
*Entries :      100 : Total =            3169 bytes  File  Size =       1701 *
*        :          : Tree compression factor =   1.21                       *
******************************************************************************
*Br    0 :Det       : det/I                                                    *
*Entries :      100 : Total  Size=        936 bytes  File Size  =        230 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.04     *
*.............................................................................*
*Br    1 :En        : energy/F                                                 *
*Entries :      100 : Total  Size=        943 bytes  File Size  =        469 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    2 :Time      : time/F                                                   *
*Entries :      100 : Total  Size=        941 bytes  File Size  =        471 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
root[1] tree->Show (10)
======> EVENT:10
 det             = 10
 energy          = 3.10897
 Time            = 5.81155
root[2] tree->Scan ()
************************************************
*    Row   * Det.Det.d * En.En.ene * Time.Time *
************************************************
*        0 *         1 * 2.7607548 * 2.8281364 *
*        1 *        12 * 13.696406 * 2.2420666 *
*        2 *        12 * 21.884300 * 11.228475 *
*        3 *        11 * 10.673481 * 10.060612 *
*        4 *        17 * 16.964376 * 18.435609 *
*        5 *         5 * 9.2536840 * 7.2596163 *
```

TTree c.d.

Wykreślanie histogramu zmiennej (zmiennych, kombinacji zmiennych)

```
root[0] tree->Draw ("energy")
root[1] tree->Draw ("sqrt(energy)")
root[2] tree->Draw ("time:energy", "", "colz")
root[3] tree->Draw ("time:Entry$" )
```

← Przykład funkcji na zmiennej

← Rozkład 2-wymiarowy

← Entry$ to specjalna zmienna = numerowi wpisu

Wykreślanie histogramu przy zadanych warunkach

```
root[3] tree->Draw ("time", "det>14 && det<23")
```

Projekcja drzewa na histogram

```
root[4] tree->Project (
Long64_t Project(const char* hname, const char* varexp, const char* selection
= "", Option_t* option = "", Long64_t nentries = 1000000000, Long64_t
firstentry = 0)

root[4] TH1F henergy ("henergy", "", 15, 0., 30. );
root[5] tree->Project ("henergy", "energy", "det<=10" );
root[6] henergy.Draw ();
```

Cięcia (`TCut`)

```
root[1] TCut cut1 ("det<=10") , cut2 = "det>=20" ;
root[2] henergy.Reset ();
root[3] tree->Project ("henergy", "energy", cut1 || cut2 );
root[4] tree->Draw ("energy", cut1 && "Entry$ <= 50" );
```

← Można łączyć TCut z napisem

TTree c.d.

Cięcie graficzne na TH2 (TCutG)

```
$ wget www.fuw.edu.pl/~kpias/ctnp/auau_1.23AGeV_8dsts.root
$ nice root -l auau_1.23AGeV_8dsts.root

root[1] wars_tree->Draw ("dEdxToF:totmom", "totmom<1500 && dEdxToF<15.");
```

① W oknie graficznym: View → Toolbar , a następnie  .

② Wycinamy wielokąt: kliknięcie myszką definiuje nowy wierzchołek. Dwukrotne kliknięcie – koniec.

③ Otrzymujemy wskaźnik na obiekt: `TCutG*  CUTG .`

```
root[2] wars_tree->Draw ("dEdxToF:totmom", "totmom<1500 && dEdxToF<15. && CUTG")
root[3] CUTG->Draw ("same")
```

Używając metody `IsInside` możemy zbadać, czy para współrzędnych mieści się wewnątrz konturu, np:

```
root[4] CUTG->IsInside ( 500, 7 );
```

Cięcie można przemianować, a także zapisać w pliku:

```
root[5] CUTG->SetName ( "mycutg" );
root[6] TFile f ( "mycutg.root" , "recreate");  mycutg->Write();
```

Jednak jeśli chcemy pobrać je z pliku i użyć do selekcji z drzewa, należy wpierw przypisać osiom zmienne:

```
$ nice root -l auau_1.23AGeV_8dsts.root
root[0] TFile filecut ( "mycutg.root" );
root[1] TCutG* mycutg = (TCutG*) filecut->Get ("mycutg");
root[2] mycutg->SetVarX ("totmom");  mycutg->SetVarY ("dEdxToF");
root[3] wars_tree->Draw ("dEdxToF:totmom", "totmom<1500 && dEdxToF<15. && mycutg");
```

Pobranie TTree z pliku (.root) i odczyt danych z TTree:

① Podpinamy się pod drzewo:

```
TTree* t = (TTree*) f.Get ("tree");
```

② Podpinamy zmienne pod gałęzie:

```
t->SetBranchAddress ("nazwa",&zmienna);
```

③ Liczba eventów:

```
t->GetEntries();
```

④ Wczytujemy dane z event'u do zmiennych:

```
t->GetEntry (i);
```

```
int TTree_simple_read () {
  Int_t det;
  Float_t energy , time;

  TFile f ("simple.root");
  TTree* t = (TTree*) f.Get ("tree");
  t->SetBranchAddress ("Det" , &det   );
  t->SetBranchAddress ("En"  , &energy);
  t->SetBranchAddress ("Time", &time  );

  for (int i=0; i<t->GetEntries(); i++)
  {
    t->GetEntry (i);
    cout << setw( 5) << det
         << setw(12) << time
         << setw(12) << energy << endl;
  }
  return 0;
}
```

Status gałęzi

Istnieje możliwość wskazania, które gałęzie mają być analizowane, a które nie. Przed pętlą po eventach piszemy:

```
TTree::SetBranchStatus ("gałąź", status);
```

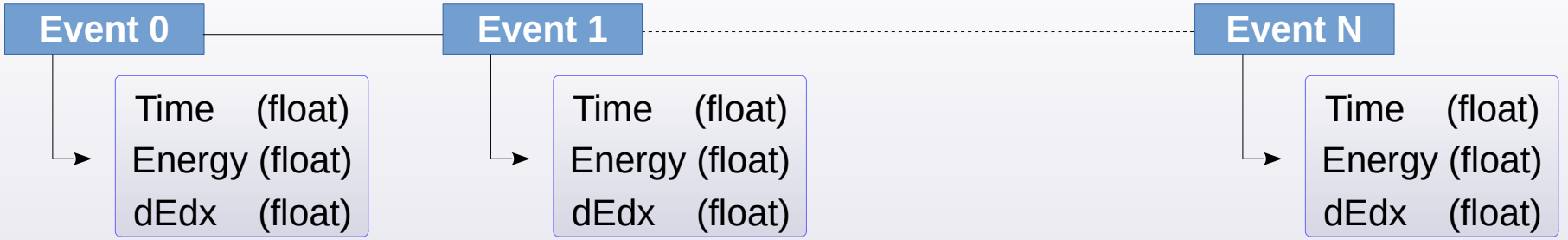
status: (0) / 1 = (nie)aktywny

"gałąź" = "*" : ustawienie dotyczy wszystkich gałęzi na raz

Wyłączenie niepotrzebnych gałęzi **skraca czas** analizy (ważne dla dużych plików!)

TNtuple (TNtupleD) Prostsze drzewa złożone tylko z *floatów* (*doubli*)

Jeszcze prostszy schemat danych (wariant floatów):



- W konstruktorze definiujemy zmienne.
- Każda zmienna stanie się gałęzią.
- Wpisując do drzewa, podajemy wartości.
- Bezpośrednie wypełnianie do 15 zmiennych

```
Fill (var1, var2, …)
```

- … lub przez tablicę:

```
Fill (Float_t* x)
```

```
int TNtuple_example () {
  Float_t energy , time, dEdx;

  TFile f ("tntuple.root", "RECREATE");
  TNtuple n ("tntuple", "My ntuple", "Time:En:dEdx" );

  TRandom3 r; r.SetSeed ();
  for (int i=0; i<100; i++) {
    time   = r.Rndm() * 20.;
    energy = r.Rndm() * 30.;
    dEdx   = r.Rndm() * 0.5;
    n.Fill (time, energy, dEdx);
  }
  n.Write();
  return 0;
}
```

Łączenie danych z plików .root o tej samej strukturze

Gdy potrzebujemy przeanalizować szereg plików z `TTree` o tej samej strukturze, można w kodzie zrobić pętlę: otwierać i-ty plik, podpinać drzewo i gałęzie, zamykać plik. Gdy jednak wynikowe histogramy składujemy w zbiorczym pliku wyjściowym, często trzeba przepinać `gDirectory`.

Istnieje alternatywa: scalenie danych wejściowych.

TChain. (Łańcuch) . Obiekt do łączenia drzew TTree.
Załóżmy, że w każdym pliku wejściowym tkwi drzewo ”T” .

① Tworzymy łańcuch:

```
TChain myChain ("T");
```

② Dodajemy kolejne pliki:

```
myChain.Add ("file1.root");
myChain.Add ("file2.root");
myChain.Add ("file3.root");
```

③ Odtąd korzystamy z obiektu `myChain` tak, jakby był wspólnym drzewem wejściowym.

Aplikacja **hadd** , wywoływana z konsoli :

```
> hadd  dane_laczne.root  dane_1.root dane_2.root ....
                          ( lub: dane_*.root )
```

Uwaga: maksymalny rozmiar pliku złączonego przez hadd to **100 GB**.
Do większych danych istnieje klasa **`TFileMerger`**. Można skorzystać z tego makra.

TTree c.d. Zapis obiektów TVectorN{N = 2, 3} / TLorentzVector w evencie:

Zapis:

```
int TTree_TVector () {

 TVector3 v3;
 TVector3* pv3 = &v3;

 TLorentzVector vL;
 TlorentzVector* pvL = &vL;

 TFile file ("TTree_TVector.root", "recreate");

 TTree* ttree = new TTree ("ttree", "ttree");
 ttree->Branch ("v3", "TVector3"       , &pv3);
 ttree->Branch ("vL", "TLorentzVector", &pvL);

 TRandom3 r; r.SetSeed (0);

 for (int evt = 0; evt < 100; evt++)
 {
   v3.SetXYZ  (r.Rndm(), r.Rndm(), r.Rndm());
   vL.SetXYZT (r.Rndm(), r.Rndm(),
               r.Rndm(), r.Rndm() );
   ttree->Fill();
 }
 ttree->Write();

 file.Close();
 return 0;
}
```

Odczyt:

```
int TTree_TVector_read () {

 TVector3 v3;
 TVector3* pv3 = &v3;

 TLorentzVector vL;
 TLorentzVector* pvL = &vL;

 TFile f ("TTree_TVector.root");

 TTree* ttree = (TTree*) f.Get ("ttree");
 ttree->SetBranchAddress ("v3", &pv3 );
 ttree->SetBranchAddress ("vL", &pvL );

 for (int evt=0; evt < ttree->GetEntries(); evt++)
 {
   ttree->GetEvent (evt);

   cout << "[" << evt << "]: ["
        << fixed << setprecision (3) <<
        << v3[0] <<" : "<< v3[1] << " : "
        << v3[2] << "]" << "\t";

   cout << "["   << vL[0] << " : " << vL[1]
        << " : " << vL[2] << " : " << vL[3]
        << "]\n";
 }
 f.Close();

 return 0;
}
```

Uwaga: działają metody klas `TVector3` i `TLorentzVector`, np: `tree->Draw ("v3.Mag()")`

TTree c.d. Eventy o zmiennej liczbie cząstek (sposób najprostszy)

Zapis:

```
int TTree_EventManyParticles () {
 Int_t Npart;
 Int_t det[500];
 Float_t energy[500] , time[500];

 TFile f ("manyparticles.root", "RECREATE");
 TTree t ("tree", "My tree");
 t.Branch ("Npart", &Npart, "Npart/I");
 t.Branch ("Det"  , det   , "det[Npart]/I");
 t.Branch ("Time" , time  , "time[Npart]/F");
 t.Branch ("En"   , energy, "energy[Npart]/F");

 TRandom3 r; r.SetSeed ();
 for (int ievt=0; ievt < 100 ; ievt++)
 {
  Npart = r.Integer(6);
  cout << "Event " << ievt
       << " has " << Npart << " particles.\n";
  for (int ipart=0; ipart<Npart; ipart++)
  {
    det   [ipart] = r.Integer (24);
    time  [ipart] = r.Rndm() * 20.;
    energy[ipart] = r.Rndm() * 30.;
    cout << setw(10) << det   [ipart]
         << setw(12) << time  [ipart]
         << setw(12) << energy[ipart] << endl;
  }
  t.Fill ();
 }
 t.Write();
 return 0;
}
```

Odczyt:

```
int TTree_EventManyParticles_read () {
 Int_t Npart;
 Int_t det[500];
 Float_t energy[500] , time[500];

 TFile f ("manyparticles.root", "READ");
 TTree* t = (TTree*) f.Get ("tree");
 t->SetBranchAddress ("Npart", &Npart );
 t->SetBranchAddress ("Det"  , det   );
 t->SetBranchAddress ("Time" , time  );
 t->SetBranchAddress ("En"   , energy );

 cout << "* This tree has "
      << t->GetEntries() << " entries.\n\n";

 for (int ievt=0; ievt<t->GetEntries(); ievt++)
 {
  t->GetEntry (ievt);
  cout << "* Event " << ievt
       << " has " << Npart << " particles:\n";

  for (int ipart=0; ipart<Npart; ipart++)
  {
   cout << setw(10) << det   [ipart]
        << setw(12) << time  [ipart]
        << setw(12) << energy[ipart] << endl;
  }
 }
 return 0;
}
```

Mankament: należy ustalić maksymalny rozmiar tablic (tu: 500). Dynamiczna alokacja nie działa...

TTree c.d. Obiekty własnej klasy w drzewie

Sposoby implementacji zmieniały się z wersjami.
Proponowany sposób dla Root 5,6 : poprzez ACLiC. Poniżej – kod dla minimalnego obiektu.

1. Tworzymy plik nagłówkowy `myClass.h`

```
#ifndef __myClass__
#define __myClass__
#include "TObject.h"

class myClass : public TObject {
 public:
  Int_t det;        // det
  Double_t ToF;     // ToF
  Double_t Energy; // Energy

  myClass() { det = 0;
              ToF = 0.; Energy = 0.; }

 //  Deklaracje innych naszych metod

  ClassDef (myClass,1)  // My simple class
};
#endif
```

b/średnika

2. Tworzymy kod źródłowy klasy `myClass.cxx` :

```
#include <iostream.h>
#include <myClass.h>

ClassImp(myClass)

// Implementacje innych naszych metod
```

b/średnika

→ Klasa musi dziedziczyć po `TObject`.
→ Musi mieć konstruktor `()`.
→ `ClassDef` oraz `ClassImp` to makra preprocesora, które wklejają w to miejsce dodatkowe metody, m.in. umożliwiające zapis obiektu w drzewie (`::Streamer`) oraz utworzenie dokumentacji.

TTree c.d. Obiekty własnej klasy w drzewie

3. Kodujemy tworzenie drzewa, które do każdego eventu wpisze 1 obiekt `myClass`.

```
#ifdef __CINT__
#else
#include "myClass.h"
#endif

int TTree_myObject ()
{
  if (!TClass::GetDict("myClass"))
    gROOT->ProcessLine (".L myClass.cxx+");

  TRandom3 r; r.SetSeed ();
  myClass* myObj = new myClass ();

  TFile f ("myobjs.root", "recreate");
  TTree* t = new TTree ("tree", "My Tree");
  t->Branch ("myObj", &myObj, 8000, 0);

  for (int evt = 0; evt < 100; evt++) {
    myObj->det    = r.Integer (24) ;
    myObj->ToF    = r.Rndm() * 20. ;
    myObj->Energy = r.Rndm() * 30. ;
    t->Fill();
  }
  t->Write();
  t->Print();
  f.Close();
  return 0;
}
```

Uzgadnia wersje dla Root 5 i 6

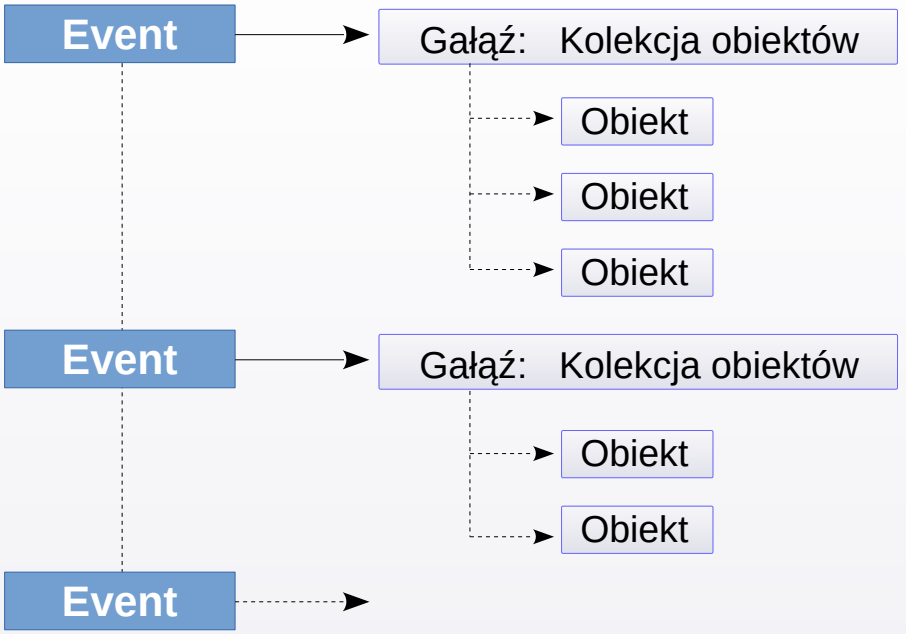
4. Kodujemy też odczyt takiego drzewa.

```
#include "myClass.h"

int TTree_myObject_read ()
{
 myClass* myObj = new myClass ;

 TFile f ("myobjs.root");
 TTree* t = (TTree*) f.Get ("tree");
 t->SetBranchAddress ("myObj", &myObj);

 cout << "This tree has "
      << t->GetEntries() << " events.\n";
 for (int evt=0; evt < t->GetEntries(); evt++)
 {
  t->GetEntry (evt);

  cout << "[Event " << evt << "] :  ["
       << setw( 4) << myObj->det
       << setw(12) << myObj->ToF
       << setw(12) << myObj->Energy
       << " ]\n";
 }
 f.Close();

 return 0;
}
```

Komenda `.L` utworzy na ścieżce 2 pliki:

- `myClass_cxx.so` (skompilowany obiekt – *shared library*)
- `myClass_cxx.d` (“dependencies”; informacje dla Roota)

TTree c.d. Kolekcje (identycznych) obiektów dla eventu w drzewie



- Aby zakodować taką strukture, użyjemy obiektu klasy `TClonesArray`.

  Jest to tablica obiektów tej samej klasy. Obiekty muszą dziedziczyć po `TObject` .
  Przy tworzeniu tablicy domyślny rozmiar: 1000 obiektów.
  Rozszerzanie rozmiaru jest automatyczne, gdy tylko wstawimy obiekt na dalszej pozycji.

- Nb. ROOT posiada kilka typów tablic obiektów (tzw. `Collections`).
  Przykładowo, w `TOrdCollection` można trzymać obiekty różnych klas (dziedziczące po `TObject`) .

- Na kolejnych slajdach: przykładowe kody zapisujące i odczytujące taką “strukturę” .

TTree c.d. Kolekcje (identycznych) obiektów dla eventu w drzewie

Przykładowy kod zapisujący tablice klonów w eventach drzewa:

- Tworzymy tablicę `TClonesArray`, podając nazwę klasy obiektów. Tworzymy też wskaźnik do niej. Obecnie tablica jest pusta.

  Jeśli nie podamy rozmiaru, domyślnie utworzymy tablicę “na 1000 elementów”. Jeśli przepełnimy, rozszerzy się.

- W definicji gałęzi podajemy wskaźnik do wskaźnika tablicy.

- Czyścimy tablicę

- Tworzymy nowy obiekt klasy `myClass` i jednocześnie umieszczamy go w tablicy na kolejnej pozycji.

```
#ifdef __CINT__
#else
#include "myClass.h"
#endif

int TTree_TClonesArray ()
{
 if (!TClass::GetDict ("myClass"))
   gROOT->ProcessLine ( ".L myClass.cxx+" );

 TFile f ("clonesarray.root", "recreate");

 TClonesArray* myArrayPtr = new TClonesArray ("myClass");
 myClass* myObjectPtr;

 TTree* t = new TTree ("tree", "My Tree");
 t->Branch ("ObjClones", &myArrayPtr , 256000, 0);

 TRandom3 r;  r.SetSeed();

 for (int evt=0; evt<100; evt++) {
   myArrayPtr->Clear();
   int Npart = rand() % 6;

   cout << "Event " << evt << " has " << Npart << " particles. \n";
   for (int iPart = 0; iPart < Npart; iPart++)
   {
     myObjectPtr = (myClass*) myArrayPtr->ConstructedAt (iPart);
      myObjectPtr->det    = r.Integer (24);
      myObjectPtr->ToF    = r.Rndm();
      myObjectPtr->Energy = r.Rndm();
   }
   t->Fill();
 }
 t->Print();  t->Write();
 f.Close();
 return 0;
}
```

TTree c.d. Kolekcje (identycznych) obiektów dla eventu w drzewie

Przykładowy kod odczytujący tablice klonów z eventów drzewa:

- Tworzymy 1x `TClonesArray` poprzez wskaźnik.
- Podpinając się pod gałąź, podajemy wskaźnik do wskaźnika tablicy.
- Czyścimy tablicę przed pobraniem ev.
- Gdy pobierzemy event, wypełnia się tablica.
- Iterujemy po obiektach tablicy.

```
#include "myClass.h"

int TTree_TClonesArray_read ()
{
 TFile f ("clonesarray.root");

 TClonesArray* myArrayPtr = new TClonesArray ("myClass");

 TTree *t = (TTree*) f.Get ("tree");
 t->SetBranchAddress ("ObjClones", &myArrayPtr);

 myClass* myObjectPtr;

 for (int evt = 0 ; evt < t->GetEntries() ; evt++)
 {
  myArrayPtr->Clear();
  t->GetEvent (evt);
  int Npart = myArrayPtr->GetEntries() ;

  cout << "\nEvent " << evt << " has "
       << Npart << " particles: \n";

  for (int iPart = 0; iPart < Npart; iPart++)
  {
   myObjectPtr = (myClass*) myArrayPtr→At (iPart);

   cout << " [" << setw (2) << myObjectPtr->det
        << ":  " << setw (9) << myObjectPtr->ToF
        << " , " << setw (9) << myObjectPtr->Energy << " ] \n";
  }
 }

 f.Close();
 return 0;
}
```

Miejsce zerowe funkcji

- ROOT posiada metody numeryczne, zapożyczone z biblioteki GSL. W skrypcie rozważymy 2 z nich.
- Szukając miejsca zerowego, wpierw wybieramy, czy metodzie wystarczy funkcja (np. bisekcja), czy potrzeba pochodnej (np. Newton).

① Tworzymy funkcję (+ ew. pochodną)

② Funkcję (+ ew. pochodną) wstawiamy do specjalnego obiektu: “wrappera”.

③ Tworzymy narzędzie `RootFinder`, określając typ metody numerycznej. Użyjemy metod: bisekcji i Newtona. Zestaw metod dostępny jest tutaj.

④ Łączymy narzędzie z funkcją(-ami).

⑤ Polecamy znaleźć miejsce zerowe.

⑥ Wynik podaje metoda `Root()`.

```
#include <Math/RootFinderAlgorithms.h>
#include <Math/RootFinder.h>
#include <Math/Functor.h>

using namespace ROOT::Math;

double myfunc (double x) {
  return 3*x - 10;
}

double myfunc_deriv (double x) {
  return 3 ;
}

void macro_RootFinder ()
{
  Functor1D f ( &myfunc );
  RootFinder k ( RootFinder::kGSL_BISECTION ) ;
  k.SetFunction (f, 1, 10);
  k.Solve ();
  cout << "Root via bisection: " << k.Root()
       << endl;

  GradFunctor1D g ( &myfunc , &myfunc_deriv );
  k.SetMethod ( RootFinder::kGSL_NEWTON );
  k.SetFunction ( g , 4. );
  k.Solve ();
  cout << "Root via Newton   : " << k.Root ()
       << endl;
}
```

Interpolacja pomiędzy punktami

- Dostępne narzędzie **ROOT::Math::Interpolator**, zapożyczone z biblioteki GSL.

- Kroki działania:

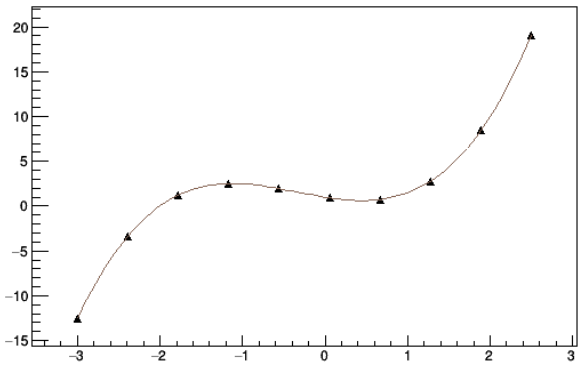
① Mamy punkty w postaci tablic `double*` lub `vector<double>`

② Tworzymy obiekt Interpolator, podając typ interpolacji:

- kLINEAR
- kPOLYNOMIAL
- kCSPLINE
- kCSPLINE_PERIODIC
- kAKIMA
- kAKIMA_PERIODIC

③ Podajemy dane: metoda `SetData`

④ Wartości funkcji interpolacyjnych są dostępne "od ręki": metoda `Eval`




```
#include <Math/Interpolator.h>

void macro_interpolation ()
{
  float xmin = -3, xmax = 2.5;
  Int_t Ndata = 10;
  double xi[Ndata], yi[Ndata];

  TF1* funPoly = new TF1 ("fp",
       "[0]+[1]*x+[2]*x^2+[3]*x^3", xmin, xmax);
  funPoly->SetParameters (1, -1.5, 1, 1);

  for (int i = 0; i < Ndata; i++) {
   xi[i]= i * (xmax - xmin) / (Ndata-1) + xmin;
   yi[i]= funPoly->Eval ( xi[i] );
  }

  ROOT::Math::Interpolator inter ( Ndata ,
        ROOT::Math::Interpolation::kPOLYNOMIAL);

  inter.SetData (Ndata, xi, yi);

  int Nprob = 100;
  double Xint[Nprob], Yint[Nprob];

  for (int i = 0; i < Nprob; ++i ) {
    Xint[i] = i*(xmax-xmin)/(Nprob-1.) + xmin;
    Yint[i] = inter.Eval ( Xprob[i] );
  }

  TGraph* gf = new TGraph (Ndata, xi, yi);
  gf->Draw ("AP");

  TGraph* gi = new TGraph (Nprob, Xprob, Yinter);
  gi->Draw ("SAME L");
}
```

Kompilacja kodu C++ z obiektami ROOT'a

Potrzebne kroki

1. Powinien być to "porządny", kompilowalny kod. Np. powinien posiadać funkcję `main`.

2. W kodzie musimy załączyć wszystkie biblioteki dla wykorzystanych obiektów ROOT'a, np:

   ```
   #include "TH1F.h"
   ```

3. Jeżeli wykorzystujemy grafikę, należy dodać interfejs graficzny **TRint**. W tym celu,

   - załączamy bibliotekę `TRint.h`
   - Funkcję `main` deklarujemy z argumentami wejścia:

     ```
     int main  (int argc, char* argv[] )
     ```

   - W ciele funkcji `main` tworzymy obiekt klasy `TRint`

     ```
     TRint myRint ("myRint", &argc, argv);
     ```

4. Kompilacja z typowymi narzędziami – poprzez:

   ```
   g++ kod.C `root-config --cflags --libs`
   ```

   Dodatkowe biblioteki dopisujemy na końcu, np. `-lMathMore` dla `Root::Math` lub `-lSpectrum` dla `TSpectrum`.

Przykładowy kod w C++ : fit TF1 do TGraph. Kompilujemy go j. w.

Cstandalone_fitTGraphErrors.C

```
#include "TF1.h"
#include "TGraphErrors.h"
#include "TMath.h"

#include "TRint.h"       // interfejs do grafiki
#include "TCanvas.h"

using namespace std;

Double_t myFun (Double_t* xarg, Double_t* par)
{
  Double_t x = xarg[0] , result = 0.;

  for (int st=0; st<=3; st++)
    result += par[st] * TMath::Power (x, st);

  return result;
}


int main  (int argc, char* argv[])
{
 TRint myRint ("myRint", &argc, argv);
 TCanvas* can1 = new TCanvas ("can1",
                              "can1", 600, 400);

 TGraphErrors gr ("dataPoints.txt");
 gr.SetTitle ();

 TF1 fun ("fun", myFun , -3, 5, 4);
 fun.SetParameters (-1. , 1. , -6. , 1.);
 gr.Fit ( &fun , "" );
 gr.Draw ("AP");

 can1->Update();
 cin.ignore();
 return 0;
}
```

Kompilacja przez *make*

W systemie Linux instalacja wielu aplikacji z kodu źródłowego dokonywana jest przez `make`.
Zadaniem `make` jest kompilacja i, jeśli potrzeba, zlinkowanie całości.

Makro do `make` dla kodu z poprzedniej strony.
Nie wykonuje linkowania, ale ma gotowe opcje w razie potrzeby.

makefile

```
CC=g++
CFLAGS=`root-config --cflags --libs`
LDFLAGS=`root-config --glibs`

SOURCE=Cstandalone_fitTGraphErrors.C
TARGET=Cviamake_fitTGraphErrors

Cviamake_fitTGraphErrors: $(SOURCE)
        $(CC) -o $(TARGET) $(SOURCE) $(CFLAGS)

clean:
        rm -f ./*~ ./*.o ./Cviamake_fitTGraphErrors
```

Unifikacja kodu

Istnieją rozwiązania na wspólny kod, obsługujący 2 warianty wykonania:

① jako kod kompilowalny np. przez g++ (z użyciem flag ROOT'a)

② jako makro w sesji ROOT'a,

Jednym z rozwiązań jest użycie komend preprocesora #if defined.

Kod demonstracyjny pokazuje też,
– jak obsłużyć argumenty wejścia
– jak korzystać z bibliotek.

```
#if defined __CINT__ || defined __CLING__

int macro_cprogram_unifier (int InputValue = 123)  {
  cout << "\n Hello, I am being interpretted." << endl;

#else

#include "TMath.h"
#include <iostream>
#include <iomanip>
using namespace std;

int main (int argc, char* argv[])  {
  cout << "\n Hello, I was compiled." << endl;
  int InputValue = (argc > 1) ? atoi (argv[1]) : 123;

#endif

  cout << "\n Okay, and this is the common portion of code.";
  cout << "\n TMath::Pi() = " << setprecision (18) << TMath::Pi();
  cout << "\n Input value (default: 123) = " << InputValue;
  cout << "\n\n";
  return 0;
}
```