Pracownia Specjalistyczna

Wprowadzenie do pakietu ROOT
Część 1

- informacje organizacyjne
- C/C++ i Linuks – przypomnienie
- ROOT – informacje wstępne
- pliki odczyt i zapis
- histogramy

dr Katarzyna Rusiecka
Zakład Fizyki Hadronów IF UJ
katarzyna.rusiecka@uj.edu.pl

Plan zajęć

1. Wprowadzenie
- C/C++ i Linuks – przypomnienie
- sesja robocza w środowisku ROOT
- standardy programowania
- konfiguracja środowiska roboczego
- GUI
- zmienne globalne
- gdzie szukać pomocy
- standardowe wejście/wyjście do pliku – przypomnienie
- pliki ROOT – zapisywanie i odczytywanie

2. Wizualizacja danych – histogramy i wykresy
- rodzaje histogramów
- tworzenie i wypełnianie histogramów
- operacje na histogramach
- rodzaje wykresów
- tworzenie i wypełnianie wykresów

3. Funkcje
- rodzaje funkcji
- tworzenie funkcji
- fitowanie

4. Drzewa
- architektura drzew
- zapisywanie danych do drzewa
- odczytywanie danych z drzewa

Linuks – przypomnienie podstawowych komend

```
$ pwd  # pokaż ścieżkę do bieżącego katalogu
$ ls  # wyświetl zawartość bieżącego katalogu
$ ls -l  # wyświetl zawartość bieżącego katalogu ze szczegółami
$ ls -a  # wyświetl zawartość bieżącego katalogu wraz z plikami ukrytymi
$ cd katalog  # wejdź do katalogu katalog
$ cd ..  # przejdź do katalogu macierzystego
$ cd  # przejdź do katalogu domowego, równoważnie: cd ~
$ mkdir katalog  # utwórz nowy katalog o nazwie katalog
$ touch plik1  # utwórz nowy plik o nazwie plik1
$ cat plik1  # wyświetl zawartość pliku plik1
$ less plik1  # wyswietl zawartość pliku plik1 we fragmentach
$ mv plik1 plik2  # zmień nazwę pliku plik1 na plik2
$ mv plik2 katalog/  # przenieś plik1 do katalogu katalog
$ cp plik2 plik3  # utwórz kopię pliku plik2 o nazwie plik3
$ cp -r katalog nowy_katalog  # utwórz kopię katalogu katalog o nazwie nowy_katalog
$ rm plik3  # usuń plik plik3
$ rm -r nowy_katalog  # usuń katalog nowy_katalog
```

Pętla `for` - przypomnienie

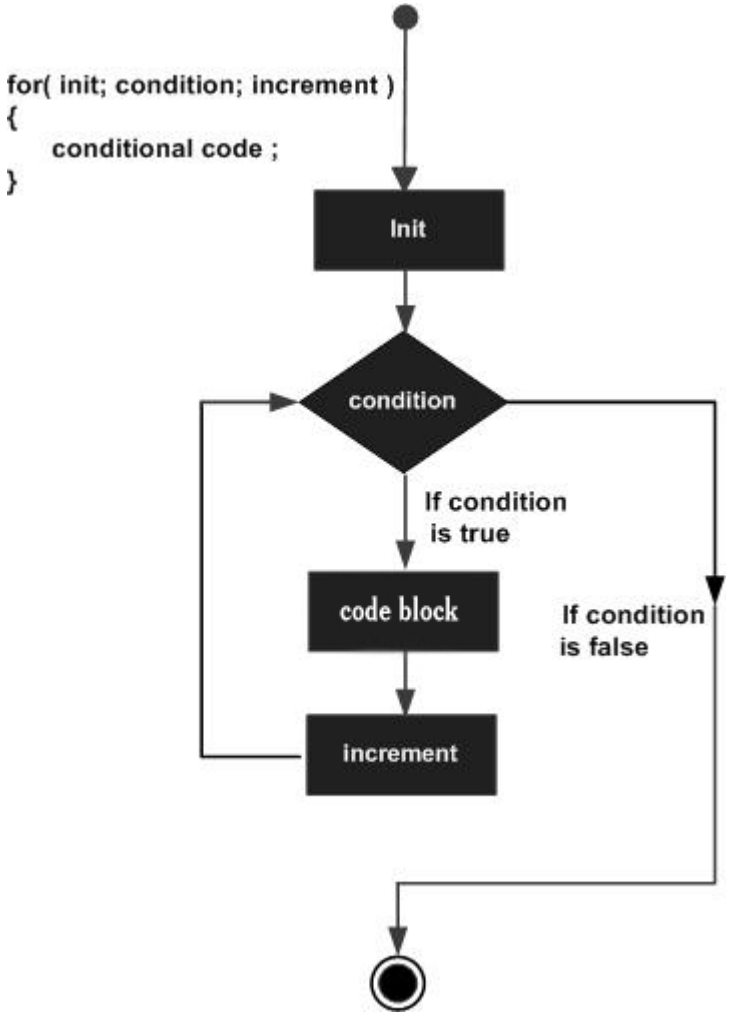
Pętla `for` pozwala na wykonanie pewnej części kodu określoną ilość razy

Składnia

```
for(init; condition; increment){
  statement(s);
}
```

Przykład

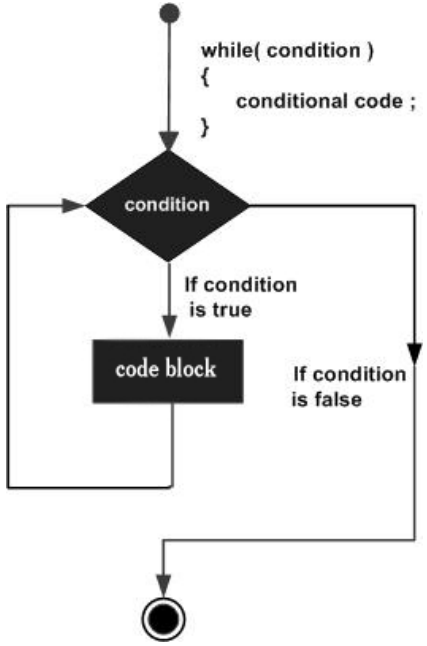
```
for(Int_t i=0; i<10; i++){
  Double_t x = 5*i - 2.5;
  cout << i << "\t" << x << endl;
}
```



https://www.tutorialspoint.com/cplusplus/cpp_for_loop.htm

Pętle `while` i `do… while…` - przypomnienie

Pętla `while` pozwala na wykonanie części kodu, dopóki podany warunek jest spełniony

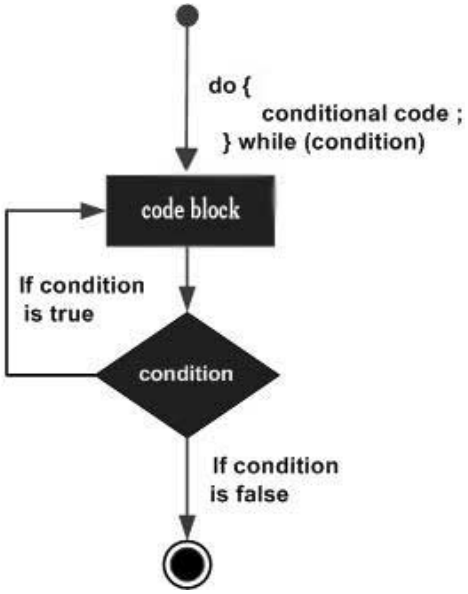


Składnia

```
while(condition){
  statement(s);
}
```

Przykład

```
Int_t i = 0;
while(i<10){
  Double_t x = i*i*3 + i*6 – 2;
  cout << i << "\t" << x << endl;
  i++;
}
```



Pętla `do… while…` jest podobna do pętli `while`. Różnica plega na tym, że w przypadku pętli `do… while…` mamy pewność, że kod wykona się przynajmniej 1 raz.

Składnia

```
do{
  statement(s);
}
while(condition);
```

Przykład

```
Int_t i = 1;
do{
  Double_t x = i*4 + 3.5;
  cout << i << "\t" << x << endl;
  i++;
}
while(x<100);
```

https://www.tutorialspoint.com/cplusplus/cpp_while_loop.htm
https://www.tutorialspoint.com/cplusplus/cpp_do_while_loop.htm

Instrukcja warunkowa `if… else…` - przypomnienie

Instrukcja warunkowa `if… else…` pozwala na wykonanie bloku kodu, jeśli podane wyrażenie logiczne jest prawdziwe. Jeśli podane wyrażenie jest fałszywe, wykonany zostanie inny blok kodu.
Po bloku `if…` nie musi nasępować blok `else…`
Jeśli chcesz sprawdzić kilka wyrażeń logicznych możesz dodać dodatkowy blok `else if(boolean expression).`

Składnia

```
if(boolean expression){
  statement(s) that will be executed if expression is true;
} else{
  statement(s) that will be executed if expression is false;
}
```

Przykład 1

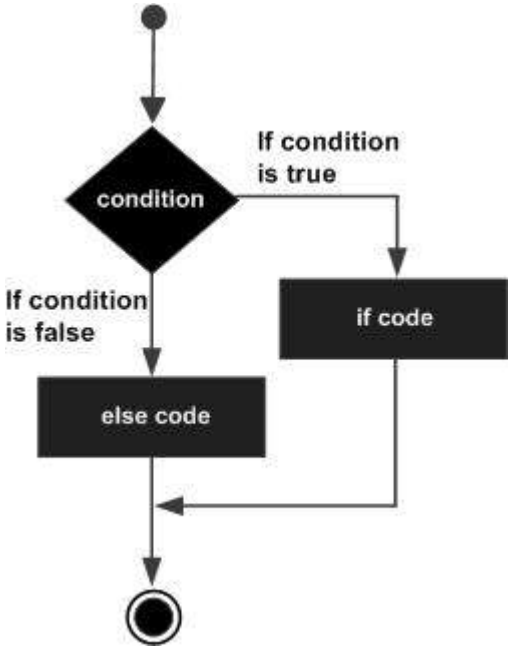
```
Double_t delta = stop - start;

if(delta == 0){
  cout << "delta equals 0!" << endl;
} else{
  cout << "delta does not equal 0!" << endl;
}
```

Przykład 2

```
Double_t delta = stop - start;

if(delta == 0){
  cout << "delta equals 0!" << endl;
} else if(delta<0){
  cout << "delta is less than 0!" << endl;
} else{
  cout << "delta is greater than 0!" << endl;
}
```



→ jeśli instrukcja warunkowa składa się z wielu bloków `else if…` warto zastąpić ją konstrukcją `switch… case…`

https://www.tutorialspoint.com/cplusplus/cpp_if_else_statement.htm

Sesja robocza w środowisku ROOT

Aby uruchomić ROOT należy wpisać w terminalu:

```
$ root
```

Opcje:
-b – batch mode, bez grafiki
-n – zignoruj rootlogon.C i rootlogoff.C
-q – zakończ sesję po wykonaniu makra
~~-l – nie pokazuj ekranu powitalnego~~
dir – jeśli dir jest istniejącym katalogiem, wejdź do niego przed uruchomieniem ROOT'a
--web=off – najnowszych wersjach, otwieraj TBrowser, zamiast RBrowser

Przykłady:

```
$ root --web=off
```

→ Przy wywołaniu GUI wyświetli się tradycyjny TBrowser

```
$ root -b -q macro.C > logfile.txt
```

→ ROOT będzie działał w trybie “batch”, “macro.C” zostanie wykonane, a output zostanie przekierowany do pliku “logfile.txt”. Po wykonaniu makra sesja zostanie zamknięta.

```
$ root 'macro.C(argument)'
```

→ Makro “macro.C” zostanie załadowane i wykonane z podanymi argumentami.

Ładowanie makra:

```
root [0] .L macro.C
```

Otwieranie GUI:

```
root [0] new TBrowser
```

Ładowanie i wykonywanie makra:

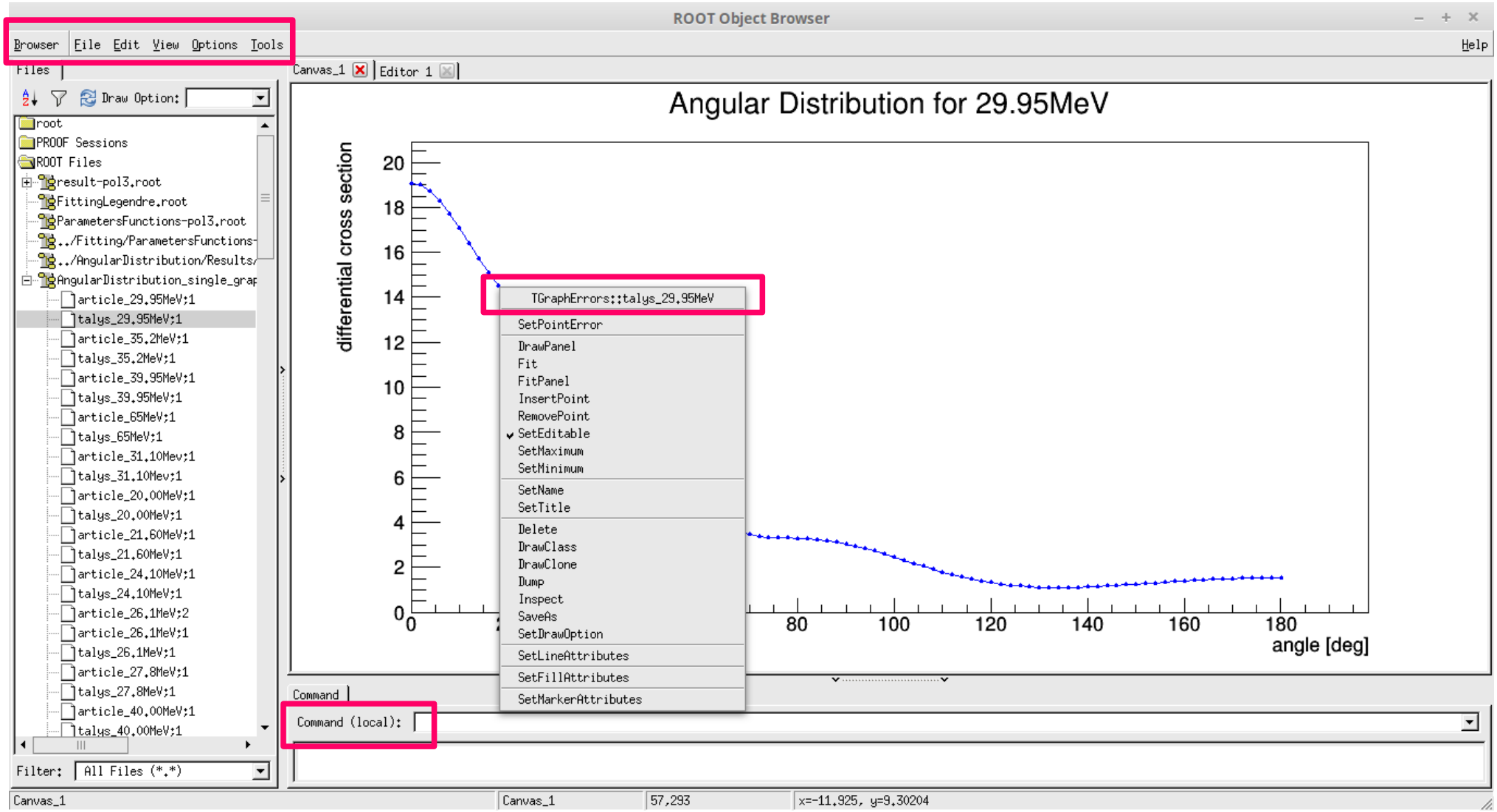
```
root [0] .x macro.C
```

Wyjście z sesji:

```
root [0] .q
```

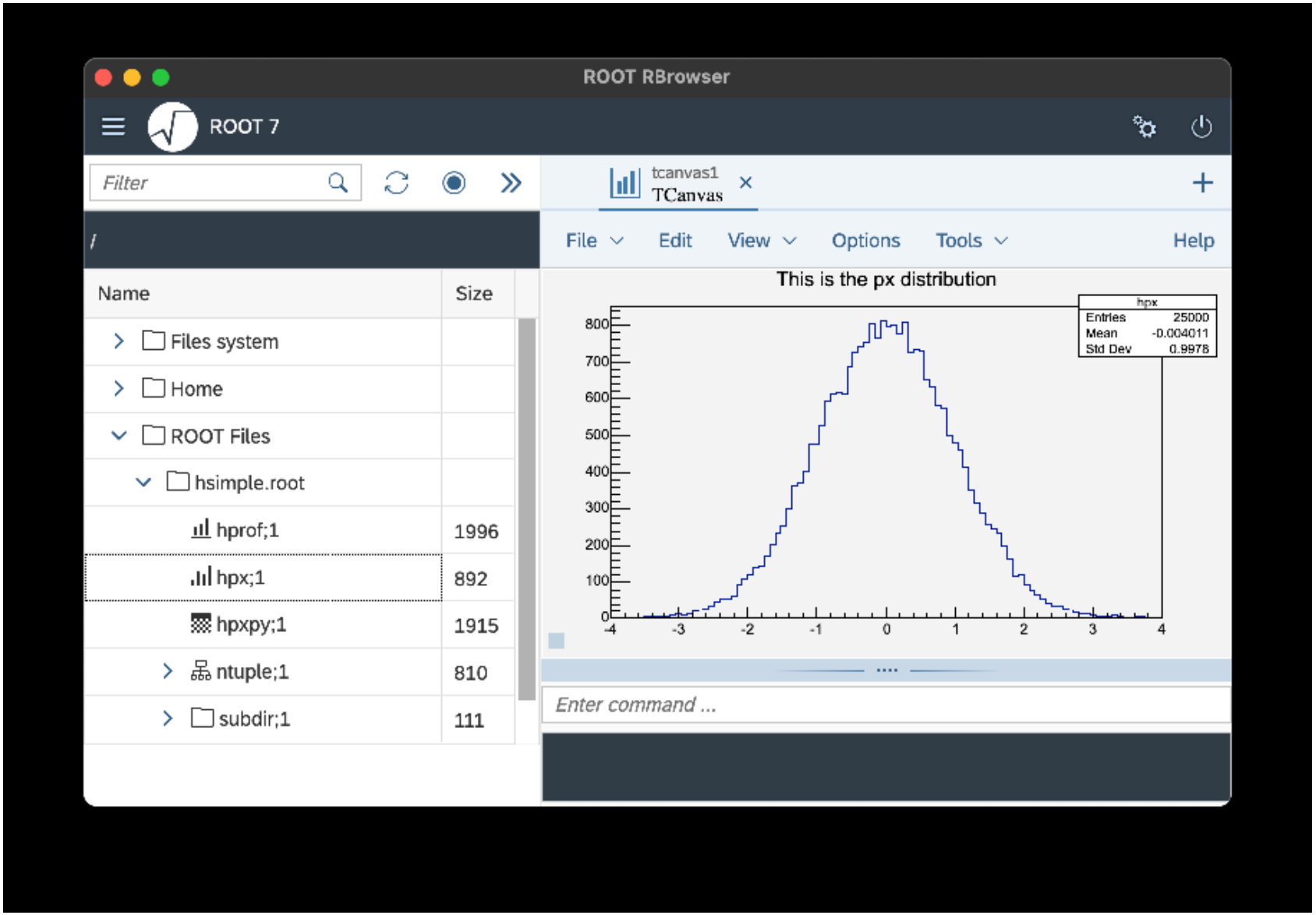
Sesja robocza w środowisku ROOT

Zadanie:
Za pomocą **TBrowser lub RBrowser** otwórz przykładowy plik **example.root** i zapoznaj się z funkcjami i możliwościami jakie daje GUI



Sesja robocza w środowisku ROOT

Zadanie:
Za pomocą **TBrowser lub RBrowser** otwórz przykładowy plik **example.root** i zapoznaj się z funkcjami i możliwościami jakie daje GUI



https://root.cern/doc/master/classROOT_1_1Experimental_1_1RBrowser.html

Standardy kodowania

Standardy ROOT'a

- Nazwy klas zaczynają się od `T`, np. `TCanvas`, `TLine`, `TGraph`
- Nazwy typów kończą się `_t`, np: `Int_t`, `Bool_t`
- Nazwy funkcji zaczynają się z wielkiej litery, np. `Loop()`, `Fill()`, `Draw()`
- Stałe zaczynają się od `k`, np. `kRed`, `kTRUE`
- Zmienne globalne zaczynają się od `g`, np. `gPad`, `gFile`
- Funkcje typu “getters” i “setters” zaczynają się od `Get` lub `Set`, np. `GetEntry()`, `SetBinContent()`

Typy zmiennych

Nazwa	Opis	Rozmiar [bajty]	Nazwa	Opis	Rozmiar [bajty]
`Char_t`	**signed char**	**1**	`Long64_t`	signed long integer	8
`UChar_t`	unsigned char	1	`ULong64_t`	unsigned long integer	8
`Short_t`	signed short integer	2	**`Float_t`**	**float**	**4**
`UShort_t`	unsigned short integer	2	**`Double_t`**	**float**	**8**
`Int_t`	**signed integer**	**2**	**`Bool_t`**	**boolean**	**1**
`UInt_t`	unsigned integer	2			

Wskazówki

- Trzymaj się standardów programowania zdefiniowanych w środowisku ROOT
- Zawsze nadawaj swoim zmiennym nazwy, które coś znaczą, np. `Double_t width` zamiast `Double_t asdfghj`
- Używaj nazw, które sugerują typ zmiennej (klase obiektu), np. `hGauss` – histogram przedstawiający rozkład Gaussa
- Unikaj powtórzeń w kodzie
- Pamiętaj o indentacji w kodzie
- Pisz komentarze

Przykład 1

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

Bool_t CalcCelsius(Double_t fahr = 100.0){

 cout << "This function recalculates temperature from Fahrenheit degrees to Celsius degrees" << endl;
 Double_t cels = (5./9.)*(fahr - 32.);  //formula to recalculate degF to degC
 cout << fahr << " deg F = " << cels << " deg C" << endl;
 return kTRUE;
}

Bool_t CalcFahrenheit(Double_t cels = 28.0){

 cout << "This function recalculates temperature from Celsius degrees to Fahrenheit degrees" << endl;
 Double_t fahr = 32. + (9./5.)*cels;    //formula to recalculate degC to degF
 cout << cels << " deg C = " << fahr << " deg F" << endl;
 return kTRUE;
}
```

Makra ROOT mogą zawierać funkcje, podobnie jak programy napisane w C/C++/inne. Jednak proste makro ROOT nie wymaga funkcji `main()`. Funkcją „domyślną” jest funkcja, która nazywa się tak samo jak makro, np. `ObliczDelta.C` i `ObliczDelta()`.
Makra ROOT mogą składać się także z serii poleceń zamkniętych w nawiasach klamrowych:

```
{
     polecenia
}
```

Zadanie:
- przeanalizuj przykładowe makro CalcCelsius.C
- które z omówionych standardów kodowania zostały tu zastosowane?
- wykonaj makro korzystając z omówionych dostępnych opcji

Zmienne globalne

`gROOT` – umożliwia dostęp do wszystkich obiektów utworzonych za pomocą ROOT'a. W czasie sesji ROOT'a `gROOT` posiada kolekcje dzięki którym zarządza obiektami. Kolekcje te można pobrać za pomocą metod `gROOT→GetListOf...`. Można w ten sposób na przykład odnaleźć canvas o nazwie `c1`:

```
root[0] gROOT->GetListOfCanvases()->FindObject("c1")
```

`gSystem` – umożliwia komunikację pomiędzy sesją ROOTa a środowiskiem systemu operacyjnego. Pozwala na przykład na ładowanie dodatkowych bibliotek:

```
root[0] gSystem->Load("libMathCore.so")
```

`gFile` – wskaźnik do aktualnie otwartego pliku ROOT.

`gDirectory` – wskaźnik do bieżącego katalogu.

`gPad` – wszystkie obiekty graficzne są zawsze rysowane na aktywnym padzie. Za pomocą wskaźnika `gPad` można uzyskać dostęp do bieżącego pada, np.:

```
root[0] gPad->SetFillColor(kBlue)
```

`gRandom` – wskaźnik do bieżącego generatora liczb losowych. Domyślnie wskazuje na obiekt klasy `TRandom3`. Generator liczb losowych można dowolnie zmieniać, np.:

```
root[0] delete gRandom
root[0] gRandom = new TRandom2(0)
```

`gEnv` – zmienna globalna zawierająca informacje o ustawieniach bieżącej sesji ROOT'a .

Gdzie szukać pomocy

- ROOT Reference Guide - https://root.cern/doc/master/index.html
- ROOT Users Guide
  - Dla początkujących: https://root.cern.ch/root/htmldoc/guides/primer/ROOTPrimerLetter.pdf
  - Dla zaawansowanych: https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuideA4.pdf
- Tutoriale – dostępne online, oraz w katalogu ze źródłami ROOTa
- Forum użytkowników ROOT'a - https://root-forum.cern.ch/
- Google

Konfiguracja środowiska roboczego

- Folder roboczy
- Ustawienie ścieżki do skryptu thisroot.sh – w pliku ~/.bashrc należy dopisać linijkę:

```
source /home/pracownia/Install/root-v6-26-10/bin/thisroot.sh
```

- Skrypt `rootlogon.C` – jest zawsze ładowany i wykonywany w momencie otwarcia nowej sesji ROOT'a
- Skrypt `rootlogoff.C` – jest zawsze ładowany i wykonywany w momencie zamknięcia sesji ROOT'a
- Skrypt `rootalias.C` – jest zawsze ładowany (ale nie wykonywany) w momencie otwarcia sesji ROOT'a

Przykłady w folderze: /home/pracownia/Install/root-v6-26-20/tutorials

Zadanie:
W folderze roboczym utwórz plik o nazwie rootlogon.C. Zadaniem skryptu rootlogon ma być ładowanie następujących bibliotek:

- libHist.so
- libMathMore.so
- libMathCore.so
- libTree.so
- libPhysics.so
- libMatrix.so

W zależności od tego, czy dana biblioteka została załadowana poprawnie czy nie, wypisz na ekran odpowiedni komunikat.

Standardowe wejście/wyjście do pliku – przypomnienie

Pliki nagłówkowe, które należy dołączyć na początku kodu, jeśli korzystamy z operacji wejścia lub wyjścia:

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

- `<iostream>` → Strumień wejścia/wyjścia na ekran
- `<ifstream>` → Strumień wejścia do pliku (odczyt danych)
- `<ofstrem>` → Strumień wyjścia do pliku (zapis danych)
- `<fstream>` → Strumień wejścia/wyjścia do pliku

Przykład wypisu na ekran:

```
cout << "some text" << endl;
```

Strumień wejścia z klawiatury:

```
Int_t number;
cin >> number;
```

Tworzenie strumienia do/z pliku:

```
fstrem myfile;
myfile.open("file.txt", mode);
```

tryb	opis
ios::in	Otwórz plik do odczytu
ios::out	Otwórz plik do zapisu
ios::binary	Otwórz mod w modzie binarnym
ios::app	Dodawaj do aktualnej zawartości pliku
ios::trunc	Usuń zawartość pliku i zastąp ją nowymi danymi

Przykłady:

```
fstrem myfile;
myfile.open("logfile.txt", ios::out | ios::app);

ifstream input("data.txt");
```

```
file.is_open();
file.eof();
file.close();
```

Pliki ROOT

Konstruktor pliku ROOT

```
TFile *file = new TFile("filename.root","OPTION");
```

Opcja	Opis
NEW or CREATE	Stwórz nowy plik i otwórz go w celu zapisu do pliku. Jeśli plik już istnieje, nie zostanie otwarty.
UPDATE	Otwórz istniejący plik w celu zapisu do pliku. Jeśli plik nie istnieje, zostanie utworzony.
READ	Otwórz istniejący plik do odczytu (opcja domyślna).
RECREATE	Stwórz nowy plik. Jeśli plik już istnieje zostanie nadpisany.

Przykład tworzenia katalogu i zapisywania histogramu w pliku ROOT:

```
TDirectory *dir = file->mkdir("directory");
dir->cd();
histogram->Write();
file->cd();
```

Rysowanie

Canvas (TCanvas) to obszar (okienko), w którym rysowane są obiekty graficzne, takie jak histogramy, grafy, itp.

Tworzenie canvasu:

```
Int_t height = 500;
Int_t width = 500;
TCanvas *can = new TCanvas("name", "title", width, height);
```

- Jeśli canvas nie zostanie stworzony przed narysowaniem histogramu – zostanie on automatycznie stworzony

- W jednej sesji ROOT'a można otwierać wiele canvasów (pod warunkiem, że nazywają się inaczej)

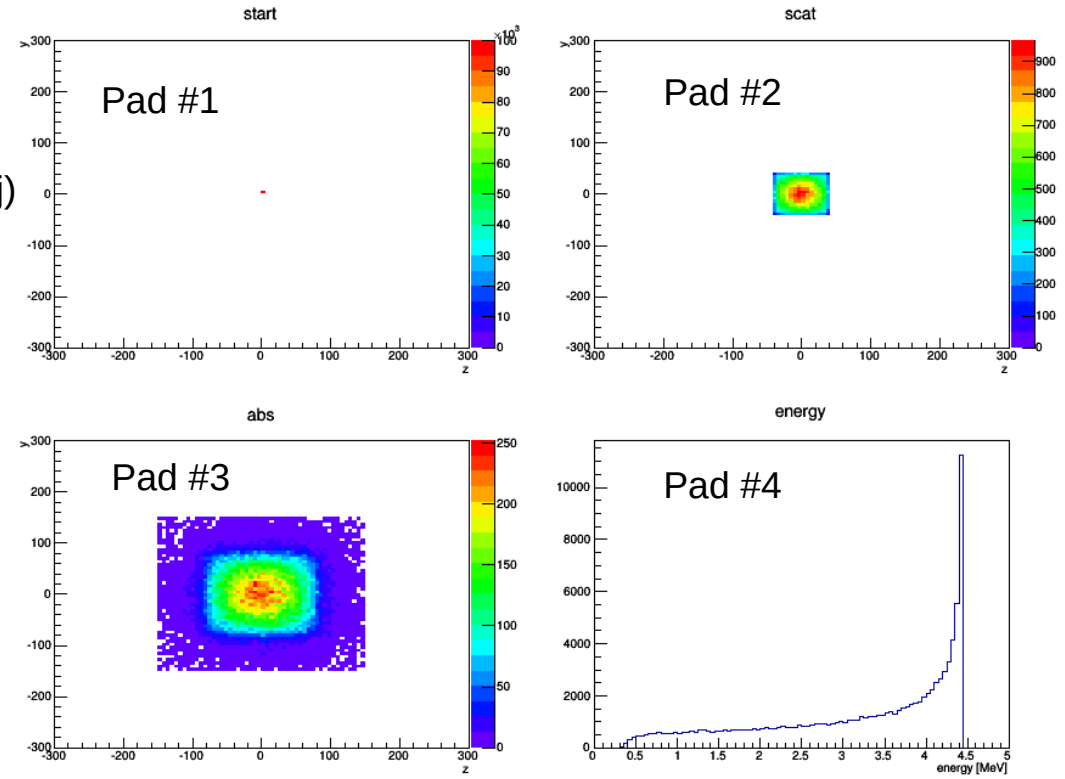
- Canvas można podzielić na mniejsze obszary - pady

Przykład podziału canvasu na pady:

```
can->Divide(2,2);
can->cd(1);
histogram->Draw();
```

- `Divide(liczba padów_w_rzędzie, liczba_padów_w_kolumnie)` lub `DivideSquare(liczba_padów)`

- Numerowanie padów zaczyna się od 1



Przykład 2

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

Bool_t DrawHistogram(void){

    TH1F *hGauss = new TH1F("gauss","gauss",50,-10,10);
    hGauss->FillRandom("gaus",5000);


    TFile *file = new TFile("histograms.root","UPDATE");

    if(!file->IsOpen()){
     cout << "Could not open output file!" << endl;
     return kFALSE;
    }

    hGauss->SetLineColor(kGreen+3);
    hGauss->SetLineWidth(2);
    hGauss->SetFillColor(kGreen-6);

    hGauss->GetXaxis()->SetTitle("x values");
    hGauss->GetYaxis()->SetTitle("y values");
    hGauss->SetTitle("Gaussian distribution");

    TCanvas *can = new TCanvas("can","can",600,600);
    can->cd();
    hGauss->Draw();

    hGauss->Write();
    file->Close();

    return kTRUE;
}
```

Załączanie plików nagłówkowych i deklaracja użycia przestrzeni nazw

Początek funkcji

Tworzenie histogramu i wypełnianie go zgodnie z rozkładem Gaussa

Tworzenie pliku ROOT

Sprawdzenie czy plik został poprawnie otworzony

Ustawianie atrybutów histogramu

Tworzenie canvasu i rysowanie histogramu

Zapisywanie do pliku i zamykanie pliku

Przykład 3

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

Bool_t DrawHistogram(void){

    (...)
     return kTRUE;
}

Bool_t GetGraphs(TString fname = ”example.root”){ ——▶ Kolejna funkcja

   TFile *f = new TFile(fname, "READ"); ——▶ Otwieranie pliku

   TGraphErrors *g = (TGraphErrors*)f->FindObjectAny("article_65MeV"); ——▶ Pobieranie
   g->Draw();                                                              wykresu z pliku

   return kTRUE;
}
```

Zadanie

- Utwórz plik WidmoCs.C
- Plik powinien zawierać funkcję o tej samej nazwie, której argumentem będzie nazwa pliku tekstowego. Ustal wartość domyślną argumentu na “Cs-137.dat”
- Wewnątrz funkcji otwórz plik tekstowy będący jej argumentem. Sprawdź czy plik został poprawnie otwarty i wypisz odpowiedni komunikat
- Odczytaj zawartość pliku. Aby upewnić się, czy dane zostały poprawnie odczytane wypisz je na ekranie

Makro to będzie punktem wyjścia do naszych kolejnych zajęć i ćwiczeń z histogramami.