Ćwiczenie 1
Pakiet ROOT – wprowadzenie cz. 1

1. Wstęp

ROOT jest darmowym pakietem ułatwiającym analizę danych. W szczególności udostępnia:

- biblioteki graficzne do przedstawiania danych (histogramy, grafy, ...),
- struktury danych (ntuple, drzewa),
- procedury do analizy danych,
- skompresowany, niezależny od systemu operacyjnego format plików z danymi.

Szczegóły instalacji: https://root.cern.ch/downloading-root

Na naszym systemie w CC1 pakiet ROOT został już zainstalowany.

2. Pakiet ROOT – pierwsze kroki

Uwaga! Przy niektórych zadaniach widnieje znak **(P)**. Oznacza podpowiedź, którą można znaleźć na ostatniej stronie.

1. W celu uruchomienia ROOTa należy wpisać w terminalu polecenie:
   `root`
2. W celu zamknięcia ROOTa należy wpisać w konsoli programu polecenie:
   `.q`
3. Początkowa grafika nie jest potrzebna i spowalnia otwieranie, dlatego w dalszej części ćwiczenia będziemy otwierać ROOTa z opcją "-l":
   `root -l`
   `.q`
4. Ze strony praktyk należy ściągnąć i zapisać w katalogu `/media/cw_ROOT_1` plik w formacie ROOT (`ROOT_cw1.root`) za pomocą:
   - przeglądarki lub
   - polecenia `wget http://atlas.ifj.edu.pl/praktyki/materialy_2016/ROOT_cw1.root`
5. Otworzyć plik programem ROOT:
   `root -l ROOT_cw1.root`
6. Wylistować zawartość pliku – w konsoli programu należy wpisać:
   `.ls`
   Pojawią się następujące informacje:
   `TFile** ROOT_cw1.root`
   `TFile* ROOT_cw1.root`
   `KEY: TTree ntuple; dane o koszulkach`
   Co znaczą wyświetlone informacje?
7. Dane (przypadki) znajdują się w drzewie nazwanym `ntuple`. Wyświetlić informacje o pierwszym przypadku:
   `ntuple->Show(0)`
   Co zostało wyświetlone?
8. Wyświetlić informację o piątym przypadku.

9. Wyświetlić informację o wszystkich przypadkach:
   `ntuple->Scan()`
   Jakie informacje zostały wyświetlone? Jest tego trochę dużo – nie będziemy oglądać wszystkich (wpisać `q`).
10. Wyświetlmy informację o pierwszych 3 przypadkach. Na logikę:
    `ntuple->Scan(2)`
    Nie działa? Jaki błąd został wyświetlony? Jaka składnia została podpowiedziana przez ROOTa?
    `ntuple->Scan(" ", " ", " ", 3)`
    `ntuple->Scan(" ", " ", " ", 3, 10)`
11. Narysujmy pierwszy histogram:
    `ntuple->Draw("l_paczek_S")`
    Pytania:
    - ile jest wejść do histogramu? .....
    - ile binów ma histogram? .....
    - w jakim zakresie rysowane są wartości? .....
    - jaka jest średnia? .....
12. Umiejętność takiego przeglądania plików jest przydatna, ale niezbyt wygodna. Dlatego często używamy interfejsu graficznego:
    `new TBrowser`
    Zadania:
    Używając interfejsu graficznego:
    - wyświetlić zawartość pliku `ROOT_cw1.root`,
    - wyświetlić zawartość drzewa `ntuple`,
    - narysować histogram zmiennej `nr_transportu`,
    - narysować histogram zmiennej `l_paczek_S`,
    - zmienić skalę na osi y na logarytmiczną **(P)**.

3. Pierwszy program do analizy danych

Praca bezpośrednio na pliku z danymi jest niepraktyczna – każde polecenia trzeba osobno wpisać lub „wyklikać". W związku z tym zwykle tworzymy makra lub programy do analizy danych.

Przykładowy program, który otwiera dane w formacie ROOTa i rysuje histogram można ściągnąć ze strony praktyk: http://atlas.ifj.edu.pl/praktyki/materialy_2016/cw1_template.cpp. Prześledźmy jego strukturę. Na początku zostały zadeklarowane potrzebne biblioteki, zarówno standardowe:

```
1 #include <iostream>
```

jak i ROOTowskie:

```
4 #include <TFile.h>
5 #include <TTree.h>
6 #include <TH1F.h> //histogramy jednowymiarowe
7 #include <TCanvas.h>
```

Program zawiera funkcję główną (`main()`) oraz wywoływaną przez nią funkcję `analiza()`:

```
53 int main(){
54   analiza();
55   return 0;
56 }
```

W funkcji `analiza()` kolejno:

- deklarujemy typ i nazwę zmiennej:

```
13    int liczba_paczek_S;
```

- deklarujemy nazwę gałęzi:

```
16    TBranch *b_l_paczek_S;
```

- tworzymy obiekt typu plik (`TFile`), podajemy ścieżkę do pliku ROOT oraz nadajemy atrybut (tu: `READ` – tylko do odczytu, istnieją też inne np. `RECREATE`, który skasuje stare dane i zapisze nowe):

```
19    TFile *f = new TFile("ROOT_cw1.root", "READ");
```

- tworzymy zmienną typu drzewo i przypisujemy jej wartość z pliku `f`:

```
21    TTree * ntuple;
22    f->GetObject("ntuple", ntuple);
```

- przypisujemy adres zmiennej `l_paczek_S`, która znajduje się w drzewie `ntuple` do utworzonej przez nas zmiennej `liczba_paczek_S`:

```
25    ntuple->SetBranchAddress("l_paczek_S", &liczba_paczek_S, &b_l_paczek_S);
```

- tworzymy zmienną `nevents`, której przypisujemy wartość równą ilości przypadków w drzewie `ntuple`:

```
28    int nevents = ntuple->GetEntries();
```

- deklarujemy histogram jednowymiarowy (klasa `TH1F`); składnia konstruktora: `(nazwa_histogramu_w_programie, opis_histogramu, liczba_binów (int), początek_pierwszego_binu (double), koniec_ostatniego_binu (double))`:

```
31    TH1F * h_liczba_paczek_S = new TH1F("h_liczba_paczek_S", "opis", 20, 0., 20.);
```

- rozpoczynamy pętlę po wszystkich przypadkach w której:
  — wczytujemy konkretny przypadek (przypisujemy odpowiednie wartości **wszystkim** zmiennym):

```
35        ntuple->GetEntry(a);
```

  — wypełniamy histogram:

```
39        h_liczba_paczek_S->Fill(liczba_paczek_S);
```

- tworzymy miejsce do rysowania histogramu (klasa `TCanvas`), rysujemy histogram oraz zapisujemy go w żądanym formacie:

```
43    TCanvas * c1 = new TCanvas();
44    c1->Clear(); //czyszczenie kanwy
45
46    h_liczba_paczek_S->Draw();
47
48    c1->SaveAs("wyk_liczba_paczek_S.eps"); //format eps
49    c1->SaveAs("wyk_liczba_paczek_S.jpg"); //format jpg
```

Sprawdźmy jak program działa. Najpierw go trzeba skompilować.

3.1. Kompilacja i uruchomienie w programie Geany

- Otworzyć program Geany:
  - Applications → Programming → Geany lub
  - kliknąć na symbol czajniczka w górnym panelu na pulpicie.
- Otworzyć plik `/media/cw1_template.cpp`.
- Skompilować i zbudować program – **kliknąć na symbol cegiełki**.
  Uwaga! Jeżeli pojawiają się błędy w kompilacji przykładu, to najprawdopodobniej należy dodać biblioteki ROOTa:
  - Build → Set Build Commands,
  - polecenie **Compile**: `g++ -Wall -std=c++11 -I /home/student/root-6.06.06/include -c ''%f''`,
  - polecenie **Build**: `g++ -Wall -std=c++11 -I /home/student/root-6.06.06/include -o ''%e'' ''%f'' -L /home/student/root-6.06.06/lib -lCore -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lMathCore -lThread -lMultiProc -pthread -lm -ldl -rdynamic`,
- Uruchomić program – **kliknąć na symbol trybików**.
  Uwaga! Jeżeli pojawiają się błędy związane z brakiem biblioteki `libCore.so`, to należy:
  - przejść do katalogu `/etc/ld.so.conf.d/`,
  - utworzyć plik `root.conf` (w trybie administratora; `sudo`),
  - dopisać do pliku linijkę: `/home/student/root-6.06.06/lib/`,
  - wywołać polecenie `sudo ldconfig`.

3.2. Kompilacja w konsoli

W celu skompilowania programu w konsoli należy przejść do katalogu, w którym został zapisany (`/media/cw_ROOT_1`) i wykonać polecenie:

```
g++ -Wall -std=c++11 -I `root-config --incdir` -o cw1_template cw1_template.cpp `root-config --libs`
```

Znaczenie poszczególnych elementów:

- `g++` – uruchomienie kompilatora,
- `-Wall` – włączenie informacji o wszystkich (`all`) ostrzeżeniach (warning, `-W`),
- `-std=c++11` – poinformowanie kompilatora o użyciu standardu C++11,
- `-I `root-config --incdir`` – podanie ścieżki (include, `-I`) do bibliotek ROOTa;
  proszę w konsoli wpisać polecenie `root-config --incdir` i zobaczyć, w jakim katalogu się znajdują,
- `-o cw1_template` – stworzenie pliku wykonywalnego o nazwie `cw1_template`,
- `cw1_template.cpp` – ścieżka do pliku zawierającego kod programu,
- `` `root-config --libs` `` – dołączenie potrzebnych bibliotek ROOTa;
  proszę w konsoli wpisać polecenie `root-config --libs` i zobaczyć, w jakie biblioteki są dołączane.

Uwaga! Znak ` to tzw. **grawis** – akcent ciężki (https://pl.wikipedia.org/wiki/Grawis). Znajduje się najczęściej na klawiszu ze znakiem tylda (∼).

3.3. Uruchomienie w konsoli

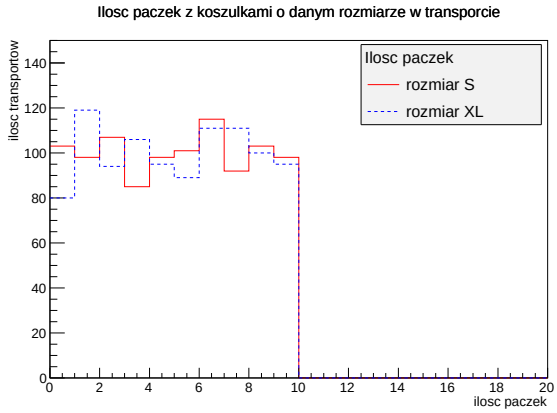
Należy wykonać polecenie:

```
./cw1_template
```

4. Zadania obowiązkowe

13. Wczytać do programu pozostałe zmienne (za wyjątkiem `typ_koszulki`) znajdujące się w pliku `ROOT_cw1.root`. W tym celu należy:
    - zadeklarować zmienne,
    - zadeklarować gałęzie,
    - przypisać odpowiednie adresy.
14. Zadeklarować histogram `h_liczba_paczek_XL` i wypełnić go wartościami `l_paczek_XL` z pliku z danymi. Liczba binów i zakres powinien być taki sam jak dla histogramu `h_liczba_paczek_S`.
15. Narysować histogram `h_liczba_paczek_XL` na tej samej kanwie, co histogram `h_liczba_paczek_S` **(P)**.
16. Zmienić kolory i typy linii histogramów: `h_liczba_paczek_S` narysować ciągłą, czerwoną linią, natomiast `h_liczba_paczek_XL` przerywaną niebieską **(P)**.
17. Usunąć z wykresu informacje statystyczne (liczba wejść, średnia, RMS) **(P)**.
18. Dodać do kanwy legendę z opisem linii **(P)**. Legenda ma zostać narysowana po prawej stronie wykresu.
19. Dodać podpisy osi oraz zmienić tytuł wykresu **(P)**:
    - tytuł: *Ilosc paczek z koszulkami o danym rozmiarze w transporcie*,
    - oś X: *ilosc paczek*,
    - os Y: *ilosc transportow*.
20. Ustawić początek (minimum) osi Y na 0, a koniec (maksimum) na 150.

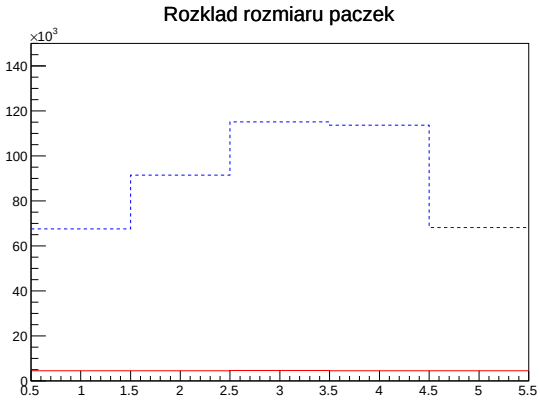
Po poprawnym wykonaniu wszystkich punktów histogram powinien wyglądać tak:

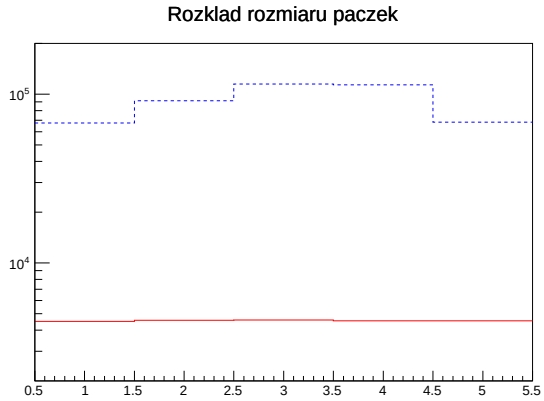


21. Zadeklarować nowy histogram `h_rozmiar` mający 5 binów w zakresie od 0.5 do 5.5.
22. Narysować rozkład pudełek z koszulkami. W tym celu dla każdego przypadku (transportu) należy wypełnić histogram `h_rozmiar` ilością paczek z danym typem koszulek. Liczba paczek z koszulkami o rozmiarze S ma trafić do pierwszego binu, M do drugiego, itd. **(P)**.
23. Narysować histogram `h_rozmiar` na pustej kanwie (nowej lub starej (po wyczyszczeniu)). Zapisać kanwę jako plik zatytułowany `wyk_rozmiar_koszulek` w formacie .eps oraz .jpg.
24. Stworzyć histogram `h_rozmiar_koszulka` pokazujący rozkład rozmiarów koszulek. Należy założyć, że ilość koszulek w paczce nie jest jednakowa, tzn. w paczkach o rozmiarze **(P)**:
    - S jest 15 koszulek,
    - M jest 20 koszulek,
    - L jest 25 koszulek,
    - XL jest 25 koszulek,
    - XXL jest 15 koszulek.

25. Dodać histogram `h_rozmiar_koszulka` do kanwy, na której narysowany został histogram `h_rozmiar_koszulka`. Zmienić kolory i typy linii obu histogramów (wybór dowolny).
26. Czy nowy wykres jest widoczny? Ustawić zakres osi y od 0 do 1.5e5.
27. Na pustej kanwie narysować te same histogramy w skali logarytmicznej. Zapisać kanwę jako plik zatytułowany `wyk_rozmiar_koszulek_log` w formacie .eps oraz .jpg **(P)**.
    Czy pojawiły się błędy?

Po poprawnym wykonaniu wszystkich punktów histogramy powinny wyglądać tak:

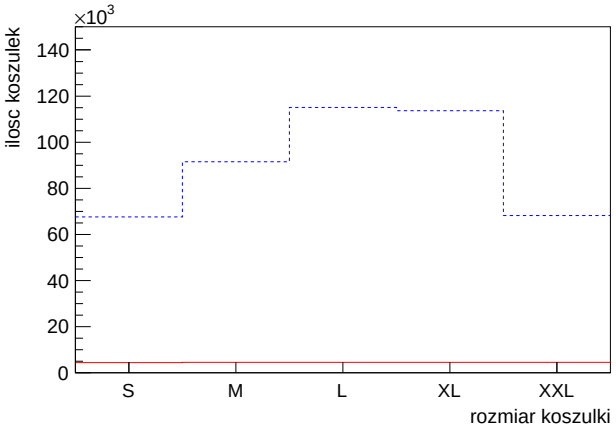


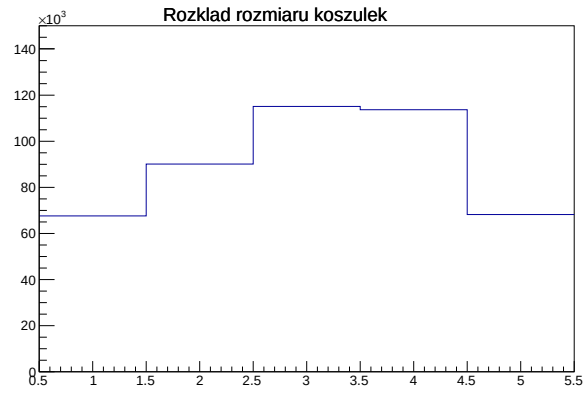


5. Zadania dodatkowe

28. Podpisać osie na histogramie `h_rozmiar_koszulka`. Wartości na osi X powinny nosić nazwy rozmiaru koszulek (zamiast numerów) **(P)**.
29. Usunąć tytuł wykresu.
30. Zmienić marginesy na kanwie (w szczególności zmniejszyć górny oraz prawy), zmienić wielkości czcionek (np. na 0.05) w podpisach osi oraz na skalach, zmienić odległość między osiami a ich podpisami (np. na 1.25) oraz między osiami a liczbami na skali (np. na 0.01) **(P)**.
31. Wczytać zmienną `typ_koszulki`, która jest wektorem typu `int` zawierającym informację o ilości koszulek danego typu w każdej z paczek w transporcie **(P)**.
32. Stworzyć histogram `h_rozmiar_koszulka_vec` na którym zostanie wyświetlona informacja o rozkładzie rozmiaru koszulek. Dane do wypełnienia histogramu wczytać ze zmiennej `typ_koszulki`.
33. Narysować histogram. Czy różni się on od histogramu `h_rozmiar_koszulka`?

Po poprawnym wykonaniu wszystkich punktów histogramy powinny wyglądać tak:





Podpowiedzi

12 Kliknąć prawym przyciskiem myszy na wykresie po lewej stronie od osi Y. W menu (zatytułowanym `TCanvas::Canvas_1`) kliknąć `SetLogy`.

15 W celu narysowania na tej samej kanwie należy użyć opcji `"same"`: `h_liczba_paczek_XL->Draw("same");`

16 Kolor danego histogramu można zmienić za pomocą funkcji: `SetLineColor(a)`, gdzie `a` jest numerem koloru. Typ linii można zmienić za pomocą funkcji `SetLineType(a)`, gdzie `a` jest numerem typu linii.

17 Wyszukać w Internecie, np. *root how to remove window with stats*. Kompilator poinformuje także o konieczności dołączenia odpowiedniej biblioteki (`TStyle.h`).

18 Biblioteka `TLegend.h`, zob. np. https://root.cern.ch/doc/master/classTLegend.html lub wyszukaj w Internecie.

19 Tytuł można zmienić za pomocą funkcji `SetTitle()`. W przypadku osi należy się do niej najpierw odwołać: `h_liczba_paczek_S->GetXaxis()->SetTitle("aa")`.

22 Istnieje kilka sposobów rozwiązania tego zadania, np.:

- (zalecany na potrzeby ćwiczenia) w każdym przypadku (transporcie) wypełnić histogram liczbą odpowiadającą rozmiarowi (np. XL = 4) tyle razy, ile było pudełek:
  `for (int b=0; b<liczba_paczek_XL; b++) h_rozmiar->Fill(4);`
- ręcznie ustawić wartość danego binu biorąc poprzednią wartość i dodając do niej odpowiednią liczbę paczek w danym przypadku (transporcie):
  `h_rozmiar->SetBinContent(4, h_rozmiar->GetBinContent(4) + liczba_paczek_XL);`

24 Istnieje kilka sposobów rozwiązania tego zadania, np.:

- (zalecany na potrzeby ćwiczenia) przy wypełnianiu histogramu nadać przypadkowi odpowiednią wagę:
  `for (int b=0; b<liczba_paczek_XL; b++) h_rozmiar_koszulka->Fill(4, 25.);`
- ustawić wartość danego binu dodając do niej liczbę paczek przemnożoną przez ilość koszulek:
  `h_rozmiar->SetBinContent(4, h_rozmiar->GetBinContent(4) + 25. * liczba_paczek_XL);`

27 Skala logarytmiczna jest właściwością obiektu `Pad`.

28 Można użyć funkcji `SetBinLabel()`.

30 Przydatne funkcje: `SetTopMargin()`, `SetTitleSize()`, `SetTitleOffset()`, `SetLabelSize()`, `SetLabelOffset()`. Pierwsza dotyczy kanwy, pozostałe odpowiedniej osi.

31 Zmienna `typ_koszulki` powinna być zadeklarowana jako wskaźnik. Dobrze jest też ją wyzerować.