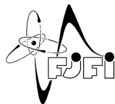
*** Break *** segmentation violation

Particle physics users workshop: from STAR to EIC

Ondřej Lomický

FJFI ČVUT v Praze

30. 1. - 2. 2. 2025



Obsah

- Spouštění ROOTu
- ROOT 5 vs ROOT 6
- Dělení chyb
- Příklady Errorů
- Segmentation violation
- Debuggování
- Příklady debuggování
- Výběr častých chyb
- StRoot třídy

Definice

debugging

/diːˈbʌgɪŋ/ -verb

Being the detective in a crime movie where you are also the murderer.

Jak správně spouštět kód (lokálně)

```
$ root --web=off -l -q VasKod.C
```

- **--web=off**: Staré grafické rozhraní ROOT
  (ne přes prohlížeč, má smysl jen pro ROOT 6)
- **-l**: Neukáže se úvodní banner s logem ROOT
- **-q**: Ukončí ROOT, jakmile kód doběhne

```
$ root --web=off -l -q VasKod.C\(5,\"input\"\)
```

- Pokud chcete dodat i vstupní parametry, nezapomeňte na lomítko \ před každou závorkou a uvozovkami

Jak správně spouštět kód (lokálně)

- Kód je ve výchozím nastavení kompilovány Clingem (ROOTovský C++ interpreter).
- Alternativně lze použít ACliC (The Automatic Compiler of Libraries in Cling), což má své výhody
  - Compiler lze optimalizovat
  - Kód lze spustit s **debugovacími symboly**
  - Kompilace je provedena jednou a příště už být nemusí (Urychlení práce)

```
$ root --web=off -b -l -q VasKod.C+

$ root --web=off -b -l -q VasKod.C++

$ root --web=off -b -l -q VasKod.C++g
```

- **+**: Spustíte kód a donutíte vytvořit sdílené knihovny *VasKod_C.so*
  Nové knihovny jsou vytvořeny pouze pokud je nějaké makro pozměněné
- **++**: Donucení rekompilace
- **++g**: Donucení kompilace s debugovacími symboly

Jak správně spouštět kód (RCF)

```
$ root -b -l -q VasKod.C

$ root4star -b -l -q VasKod.C
```

- **root4star** má dodatečné knihovny
- **-b**: Root poběží bez jakéhokoliv grafického zobrazení (Chcete na RCF, jinak trvá dlouho, než se ROOT vůbec načte)

Velké projekty na RCF, které se obvykle ukládají do složky StRoot se kompilují příkazem:

```
$ cons
```

- Příkaz **cons** je alternativa pro příkaz **make**
- Vytvoří knihovny pro daný projekt
- Potřeba použít po každé úpravě kódu

ROOT 5 vs ROOT 6 I.

- Na RCF se v současnosti používá 5.34/38
- Aktuální verze je 6.30.04, chystá se ROOT 7
- Zpětná a dopředná kompatibila by měla být do nějaké míry zaručena
- Poprvé **nebude** zaručena pro ROOT 7 :)
- Stejně narazíte na problémy, pokud budete část analýzy dělat lokálně na ROOT 6 a pak na RCF na ROOT 5 (například třída TF1)
  → Důvěřujte víc zpětné kompatibilitě než té dopředné

ROOT 5 vs ROOT 6 II.

- Na RCF existují i jiné verze ROOTu, na které může přepnout
- Vypíšete je příkazem:

```
$ ls /afs/rhic.bnl.gov/star/ROOT
```

- Pak si danou verzi můžete zvolit příkazem

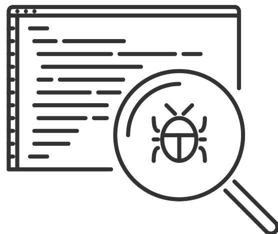
```
$ source /afs/rhic.bnl.gov/star/ROOT/6.20.08/build/bin/thisroot.
    csh
```

nebo

```
$ setup 64b
$ setup root 6.20.08
```

- Po odhlášení se vrátí zpátky výchozí verze
- Může se hodit pro spojování .root souborů (příkaz **hadd**).
- ROOT 6 je u příkazu **hadd** robustnější a má více možností

Dělení chyb



- **Error** - Chyba v kódu, kvůli které nelze kód spustit nebo zkompilovat
- **Defect** - Kód produkuje jiné výsledky, než jaké jste chtěli
- **Bug** - Chyba v napsaném kódu (Neformálně se tím označuje jakákoliv chyba)

Error I.

```
VasKod.C:6:15: error: expected ';' at end of declaration
  int a = 10
           ^
           ;
```

- Zapomenutý středník

```
VasKod.C:6:15: error: member reference type 'TCanvas *' is a
    pointer; did you mean to use '->'?
c1.cd();
~~^
  ->
```

- Špatné zacházení s ukazatelem

```
Error: Function VasKod() is not defined in current scope  :0:
*** Interpreter error recovered ***
```

- Pokud ve funkci nemáte žádnou funkce, která se jmenuje stejně jako vaše makro

Error II.

```
F.C:6:15: error: no viable conversion from 'TCanvas' to 'TH1D *'
    TH1D *h1 = *c1;
          ^     ~~~
```

- Neplatná nebo netriviální konverze (Dělá se přes **dynamic_cast** nebo **static_cast**)

```
F.C:7:9: error: use of undeclared identifier 'a'
  a = 11;
  ^
```

- Nedefinovaná proměnná

```
F.C:7:9: error: redefinition of 'a'
  int a = 11;
      ^
F.C:6:9: note: previous definition is here
  int a = 10;
      ^
```

- Redefinice

Error III.

- Na RCF při použití **cons** dostávate i upozornění na nepoužité proměnné

```
Kod.cxx:327:5: warning: unused variable 'a' [-Wunused-variable]
int a = 1;
    ^
Kod.cxx:328:9: error: 'a' was not declared in this scope
cout << a << endl;
        ^
```

- Pokud jste kreativní, chybová zpráva vám nepomůže

Error III.

- Na RCF při použití **cons** dostávate i upozornění na nepoužité proměnné

```
Kod.cxx:327:5: warning: unused variable 'a' [-Wunused-variable]
int a = 1;
    ^
Kod.cxx:328:9: error: 'a' was not declared in this scope
cout << a << endl;
        ^
```

- Pokud jste kreativní, chybová zpráva vám nepomůže
- Může to být volbou nevhodného fontu, kde např **l**, **I** a **1** či **rn** a **m** vypadají stejně
- V takovém případě je podezřelé proměnné smazat a radši napsat znova

Error IV.

2

```
326 if(true)
327 int a = 1;
328 cout << a << endl;
```

- Deklarace proměnné v **if**, **while** nebo **for** cyklu
- Tato proměnná je na konci podmínky (každého cyklu) **automaticky smazána**
- Je důležité dávat pozor, kde danou proměnnou deklarujete

Segmentation violation

- Nastává tehdy, když se program snaží přistoupit k paměti, která mu nepatří. Počítač takový program ukončí.
- Mohou nastat 3 situace:
  - ROOT vás upozorní při kompilaci (Error) a (ne)spustí se
  - ***** Break *** segmentation violation** (Fault)
  - Nepředvídatelné chování (Bug/Defekt)

```
*** Break *** segmentation violation

===========================================================
There was a crash.
This is the entire stack trace of all threads:
===========================================================
#0  0x00007ff6ee8ea3ea in __GI___wait4 (pid=30377, stat_loc=
    stat_loc
entry=0x7ffdf7bbced8, options=options
entry=0, usage=usage
...
```

- Z této zprávy nic nevyčtete

Debuggování: Googlení

- Google
  - ROOT je dost nejednoznačný název, lepší googlit chybu vždy se spojením **root cern**
  - Někdy se vyplatí hledat chybu se spojením **C++** místo ROOT
  - Používejte uvozovky ”root cern error message”, aby Google hledal přesnou formulaci
- Stackoverflow: `https://stackoverflow.com/`
- ROOT forum: `https://root-forum.cern.ch/`

Debuggování: Zakomentovávání

```
/*  //Debugging
hist->Draw();
*/  //Debugging
```

- Nejjednodušší debugging
- Řádek pomocí dvojitého lomítka: //
- Část kódu pomocí lomítka a hvězdičky: /* **zakomentovaná část** */
- Zálohujte si kód, než začnete zběsile zakomentovávat řádky
- Označujte si nějak specificky části kódu, které jste zakomentovali
- Zakomentovávání musí jít vždy zdola kódu
- Alternativou je vkládat části kódu do podmínky
  t.j. *bool DEBUG = false* a vkládat části do *if(DEBUG)*

Debuggování: Cout

```
...
cout << "1" << endl;
...
cout << "Mezivysledek: " << result << endl;
...
cout << "2" << endl;
...
```

- Dvě základní strategie:
  - Napsat do kódu různé couty a podívat se, který „zakřičí“ jako poslední
  - Vypisovat proměnné a posuzovat, jestli dávají smysl
- Obecně není příliš doporučováno
- Zpomaluje kód, pokud produkujete logy, tak mohou být větší než obvykle
- **Smažte je než pošlete joby!**
- Může se stát, že **cout** nebude správně lokalizovat místo → **cerr**

```
...
cerr << "1" << endl;
...
cerr << "2" << endl;
...
```

Debuggování: Kontrola načtení

```
TH1D *h1;
if (!h1) {
  cout << "Histogram je prazdny" << endl;
  return;
}
```

- Měli byste dělat automaticky při psaní kódu
- Lehká implementace
- Může vám ulehčit spoustu práce při debuggování

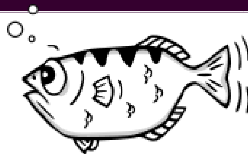
Debuggování: GDB

- Gnu Debugger (GDB) (`https://www.sourceware.org/gdb/`)
- Ve většině Linuxových distribucích je předinstalovaný
- Lze použít i na ROOT
- Na RCF jsou konfigurovány debuggovací symboly, takže tam s ROOTem funguje lépe
- Spustíte ho na svém PC následovně:

```
$ gdb --quiet --args root.exe -l -b -q Macro.C++g
(gdb) run
```

- Na RCF stačí:

```
$ gdb --quiet --args root.exe -l -b -q Macro.C
(gdb) run
```



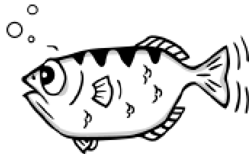
Debuggování: GDB

- Můžete vytvořit alias:

```
$ cd
$ nano .bashrc
```

- Přidejte na konec:

```
pomoc() {
  gdb --quiet -ex run --args root.exe -l -b -q "$1++g"
}
```



Debuggování: Chat GPT a GitHub Copilot

- https://chat.openai.com/
- Čím běžnější chyba, tím pravděpodobněji vám pomůže
- Nemusí vždy nabídnout nejjefektivnější řešení problému
- Bezplatná verze má omezený vstup (nemůžete poslat celý kód)
- Pokud máte dlouhý kód a vůbec z chybové hlášky není patrné, kde může být problém, tak vám nepomůže.
- Pomocí předchozích metod můžete alespoň lokalizovat část kódu, kde je problém
- GitHub Copilot chat má přístup k vašem skriptům v projektu, takže má vyšší potenciál vám pomoci

Debuggování: Eclipse IDE

- Eclipse https://www.eclipse.org/downloads/
- Robustní debugging (ROOT i GEANT4)
- Když nepomůže tohle, tak už nic
- Grafické rozhraní, není to jen příkazový řádek
- Návod: https://root.cern/blog/debuging-root-scripts-in-eclipse/
- Nevýhoda: Instalace a příprava Eclipse na debuggování je nadlouho

Příklad: Sáhnutí mimo pole

```
TH1F *hist[2];
hist[0] = new TH1F("hist1", "hist", 100, 0, 100);
hist[1] = new TH1F("hist2", "hist", 100, 0, 100);
hist[2]->Draw();
```

- Výsledek

```
$root -b -l -q SegVal.C

Processing SegVal.C...

*** Break *** segmentation violation

===========================================================
There was a crash.
This is the entire stack trace of all threads:
===========================================================
...
```

Příklad: Sáhnutí mimo pole (GDB lokálně)

```
$ gdb --quiet --args root.exe -l -b -q SegVal.C++g
Reading symbols from root.exe...
(No debugging symbols found in root.exe)
(gdb) run
Starting program: /home/ondrej/build/bin/root.exe -l -b -q SegVal.C++g
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/
    libthread_db.so.1".
...

Processing SegVal.C++g...
Info in <TUnixSystem::ACLiC>: creating shared library /home/
    ondrej/Desktop/STAR heckathon/SegV/Codes/./SegVal_C.so
...

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7f75617 in SegVal () at ../SegVal.C:15
15          hist[2]->Draw();
```

Příklad: Sáhnutí mimo pole (GDB na RCF)

```
$ gdb --quiet --args root.exe -l -b -q SegVal.C
Reading symbols from /afs/rhic.bnl.gov/star/ROOT/36/5.34.38/.
    sl73_gcc485/rootdeb/bin/root.exe...done.
(gdb) r
Starting program: /afs/rhic.bnl.gov/star/ROOT/5.34.38/.
    sl73_gcc485/rootdeb/bin/root.exe -l -b -q test.C
[Thread debugging using libthread_db enabled]
...
Processing test.C...
Error: illegal pointer to class object hist[2] 0x0 163  SegVal.C
    :15:
*** Interpreter error recovered ***

This is the end of ROOT -- Goodbye

[Inferior 1 (process 30594) exited with code 01]
```

Příklad: Sáhnutí mimo pole (Chat GPT)

- Mám takovýto kód v ROOTu, který padá:
  TH1F *hist[2];
  hist[0] = new TH1F("hist1", "hist", 100, 0, 100);
  hist[1] = new TH1F("hist2", "hist", 100, 0, 100);
  hist[2]->Draw();
  Nevíš proč?
- Ano, mám tu jednu chybu ve vašem kódu. Máte deklarováno pole hist s velikostí 2, což umožňuje pouze indexy 0 a 1. Když se snažíte vykreslit histogram hist[2], program se pokouší přistupovat k třetímu prvků pole, který neexistuje.
  Pokud chcete vykreslit druhý histogram, můžete to udělat takto:

```
hist[0] = new TH1F("hist1", "hist", 100, 0, 100);
hist[1] = new TH1F("hist2", "hist", 100, 0, 100);
hist[1]->Draw(); // Vykreslí druhý histogram
```

  Nebo pokud chcete vykreslit oba histogramy najednou, můžete použít funkci Draw("SAME"):

```
hist[0] = new TH1F("hist1", "hist", 100, 0, 100);
hist[1] = new TH1F("hist2", "hist", 100, 0, 100);
hist[0]->Draw();
hist[1]->Draw("SAME");
```

Příklad: Špatný ukazatel na intiger (defekt)

```
int *cislo = new int;
*cislo = 10;

std::cout << *cislo << std::endl;
std::cout << *(cislo+1) << std::endl;
```

- Výsledek je defekt (náhodné číslo)
- Problém, protože výstup „dává smysl“
- GDB tento problém neodhalí
- Pomůže cout nebo vykreslování do histogramů
  (pokud máte štěstí a náhodné výsledky jsou mimo očekávání)

```
$root -b -l -q kod.C
Processing kod.C...
10
21949
```

Příklad: Špatný ukazatel na string

```
std::string *str = new std::string;
*str = "Tohle je původní věta";
std::cout << *str << std::endl;
std::cout << *(str-1) << std::endl;
```

- Výsledek je náhodný
  - Náhodný **libovolně dlouhý** string
  - Segmentation violation
  - Náhodná jiná chyba
- Pokud to není důležitá část kódu, kód občas spadne a občas ne.
- GDB vám moc nepomůže (alespoň to stabilně padá), ChatGPT tohle zvládne odhalit

```
$root -b -l -q kod.C
  ???????????????}??Kcr??c?}???L?Y?????MC???????
???
??
??
?????????}??42fu?f}??%?????????f???
??????}?$3}??V<??????e?????5-f?????????????}??
????f??????????????????+????????????+??=????????       ??
```

Příklad: Otevírání souboru:

```
  TFile *file = TFile::Open("05_file_test.root", "RECREATE");
  TH1F *hist = new TH1F("hist", "hist", 100, 0, 100);

  for (int i = 0; i < 100; i++) {
    //gauss
    hist->Fill(gRandom->Gaus(50, 10));
  }

  file->Close();

  hist->Draw();
}
```

- ROOT automaticky dočasně ukládá histogramy do zrovna otevřeného souboru
- Je potřeba mu říct, kam soubor uložit

```
hist->SetDirectory(0);
```

Příklad: Počet parametrů

```
double funkce(Double_t *x, Double_t *par){
  double vysledek = par[0]*x[0]*x[0] + par[1]*x[0] + par[2];
  return vysledek;
}

void func(){
  double par[3]={1,0,1};
  TF1 *f1 = new TF1("f1",funkce,-2,2,2);
  f1->SetParameters(par);
  f1->SetParNames("a","b","c");
  f1->Draw();
  gPad->SetGrid();
}
```

- Chcete funkci $x^2 + 1$
- Vykreslená funkce je x^2, proč?
- GDB chybu neodhalí, chat GPT napsal nesmysl, ale opravil danou chybu

Příklad: Počet parametrů

```
double funkce(Double_t *x, Double_t *par){
  double vysledek = par[0]*x[0]*x[0] + par[1]*x[0] + par[2];
  return vysledek;
}

void func(){
  double par[3]={1,0,1};
  TF1 *f1 = new TF1("f1",funkce,-2,2,2);
  f1->SetParameters(par);
  f1->SetParNames("a","b","c");
  f1->Draw();
  gPad->SetGrid();
}
```

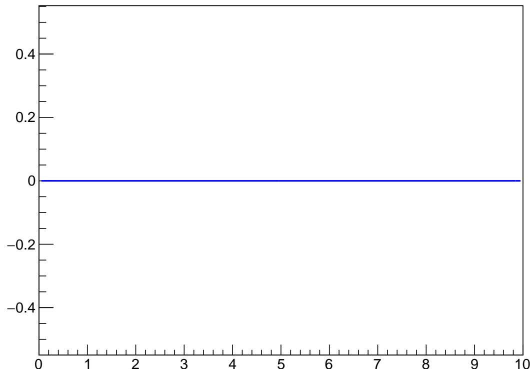
- Chcete funkci $x^2 + 1$
- Vykreslená funkce je x^2, proč?
- GDB chybu neodhalí, chat GPT napsal nesmysl, ale opravil danou chybu
- Funkce očekává jen dva parametry, takže přijme pouze dva parametry a třetí se náhodile načte

Výběr častých chyb - dělení intigerem

```
TF1 *f2 = new TF1("f2", "1/2*sin(x)", 0., 10.);
f2->SetLineColor(kBlue);
f2->Draw();
```

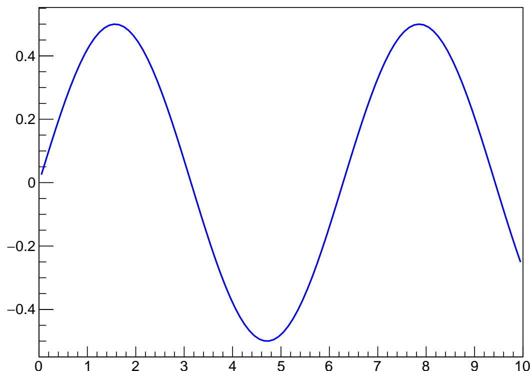
Výběr častých chyb - dělení intigerem

```
TF1 *f2 = new TF1("f2", "1/2*sin(x)", 0., 10.);
f2->SetLineColor(kBlue);
f2->Draw();
```



Výběr častých chyb - dělení intigerem

```
TF1 *f2 = new TF1("f2", "1./2*sin(x)", 0., 10.);
f2->SetLineColor(kBlue);
f2->Draw();
```



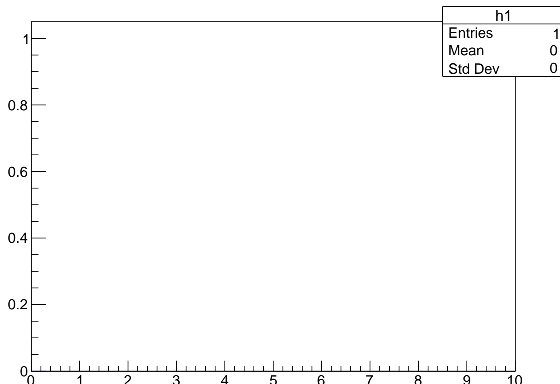
Výběr častých chyb - dělení intigerem

```
root [1] 1/2
(int) 0
root [2] 1./2
(double) 0.50000000
```

- Dělení intigerů → výsledek je taky intiger (celá dolní část směrem k nule)
- Výsledek není intiger, pokud alespoň jedno číslo intiger není (double, float)
- 1. = double, 1 = int

Výběr častých chyb - číslování od 0 a od 1

```
TH1D *h1 = new TH1D("h1", "h1", 0, 0, 10);
h1->SetBinContent(0, 1);
h1->Draw();
```



Výběr častých chyb - číslování od 0 a od 1

```
TH1D *h1 = new TH1D("h1", "h1", 0, 0, 10);
h1->SetBinContent(0, 1);
h1->Draw();
```

- Číslování je většinou od 0, biny mají číslování od 1
- bin *0* = underflow, bin *Počet binů + 1* = overflow
- $gStyle- > SetOptStat(''nemuo'');$

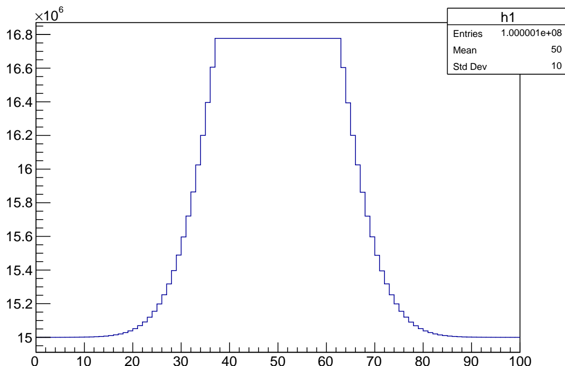
Výběr častých chyb - Mocnění symbolem ˆ

```
root [0] 1^2
(int) 3
```

- Funkce symbolu ˆ (stříška) je různá. Může fungovat jako logický XOR
- 1ˆ2 $\rightarrow$ 01ˆ10 = 11 $\rightarrow$ 3
- Vždy je lepší použít TMath::Power(*mocněnec*,*mocnitel*)

Výběr častých chyb - Přetečení floatu

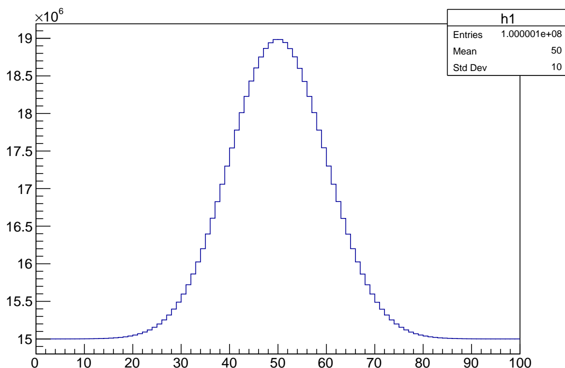
```
TH1F *h1 = new TH1F("h1", "", 100, 0, 100);
for (int i = 1; i <= 100; i++) h1->SetBinContent(i, 15000000);
for (long int i = 0; i < 100000000; i++) h1->Fill(gRandom->Gaus
    (50, 10));
```



- U čísla 16777216 jsou čísla v řádu jednotek pod jeho rozlišovací schopnost

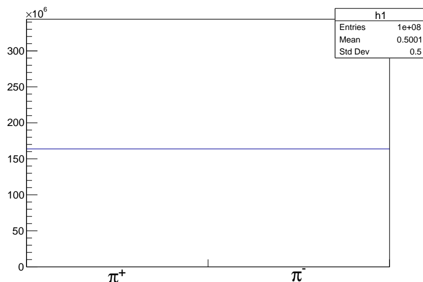
Výběr častých chyb - Přetečení floatu

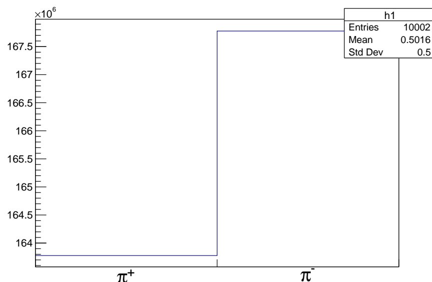
```
TH1D *h1 = new TH1D("h1", "", 100, 0, 100);
for (int i = 1; i <= 100; i++) h1->SetBinContent(i,
15000000);
for (long int i = 0; i < 100000000; i++) h1->Fill(gRandom->
Gaus(50, 10));
```



Výběr častých chyb - Přetečení floatu

- Pokud pozorovatelné (např. počet π^+ a π^-), které by měly být stejné, jsou na jednotky přesné $\rightarrow$ **podezřelé**
- Nejhorší situace je, pokud vám přeteče pouze jeden bin





Výběr častých chyb - Stejné jméno pro více objektů

```
TCanvas *c1 = new TCanvas("c1", "", 800, 600);
TCanvas *c2 = new TCanvas("c1", "", 800, 600);
```

- Zkopírování řádku bez změnění jména

```
Warning in <TCanvas::Constructor>: Deleting canvas with same
    name: c1
```

- Objeví se varování, může vést k pádu programu

StRoot třídy

- Aktuální třídy v StRoot zde:
  `https://www.star.bnl.gov/webdata/dox/html/annotated.html`
- Při přebírání kódu či používání funkcí je vždy potřeba kontrolovat „dokumentaci"
- Ve starých kódech se objevují i staré třídy
  (StThreeVector $\rightarrow$ TVector3)
- BEMC věže - ID vs softID
- Číslování centralit: 0 - nejcentrálnější/nejperifernější?
- root vs root4star - pokud nefunguje jedno, vyzkoušet druhé

Shrnutí

- Pečlivě čtěte chybové hlášky
- Postupujte co nejvíce systematicky
- Nebojte se použít AI nebo GDB, ale buďte obezřetní
- Nepoužívejte bezhlavě funkce, které neznáte, kontrolujte dokumentaci
- Nebojte se zeptat spolužáků, mohou vidět to, co vy přehlížíte
- Nepropadejte panice

Shrnutí

- Pečlivě čtěte chybové hlášky
- Postupujte co nejvíce systematicky
- Nebojte se použít AI nebo GDB, ale buďte obezřetní
- Nepoužívejte bezhlavě funkce, které neznáte, kontrolujte dokumentaci
- Nebojte se zeptat spolužáků, mohou vidět to, co vy přehlížíte
- Nepropadejte panice

Děkuji za pozornost!