Scientific C++ for MC Lectures

1. The Syntax Basics

Variables are containers. In Physics, they usually hold kinematic values.

Type	Name	Example	Physics Use Case
int	Integer	`int nEvents = 5000;`	Loop counters, number of tracks.
double	Decimal	`double mass = 91.2;`	Energy, pT, Eta, Phi.
bool	Boolean	`bool isMuon = true;`	Flags, trigger decisions.
string	Text	`string name = "data";`	Filenames, labels.

To print to the screen:

```
std::cout << "Event Number: " << i << std::endl;
```

2. Pointers & Access (The Golden Rule)

ROOT objects are heavy, so we handle them by their address (pointer).

- **The Star (*):** Declares a pointer (variable holding a memory address).
- **The Arrow (->):** Accesses functions of a pointer.
- **The Dot (.):** Accesses functions of a regular variable.

The Cheat Code: If you defined it with a *, use ->. If not, use ..

```
// Regular Variable (Stack)
TLorentzVector v1;
v1.SetPxPyPzE(1, 2, 3, 10); // USE DOT
```

```
// Pointer Variable (Heap) - 90% of ROOT Code
TH1F *hist = new TH1F("h", "title", 100, 0, 10);
hist->Fill(50); // USE ARROW
```

3. Logic & Loops (The "Analysis" Part)

Simulations happen in loops. Selections happen in `if` statements.

The Event Loop:

```
for (int i = 0; i < nEntries; i++) {
    // This code runs once for every event
    // 1. Get Entry
    // 2. Calculate
    // 3. Fill
}
```

The Cut (filter):

```
// && = AND, || = OR, ! = NOT
if (pt > 20 && abs(eta) < 2.5) {
    // Good particle, do something

} else {
    continue; // Skip to next event immediately
}
```

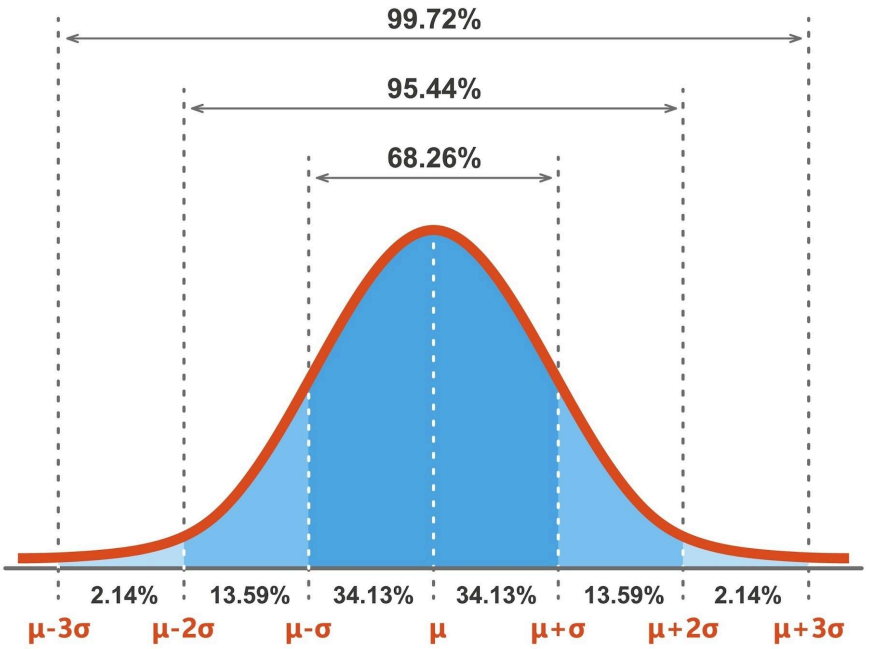
4. Standard Lists (`std::vector`)

Used when you have lists of particles.

```
#include <vector>

std::vector<double> muon_pts;

muon_pts.push_back(45.5); // Add item to end
muon_pts.push_back(20.1);

int count = muon_pts.size(); // How many? (2)
double val = muon_pts[0];    // Access first item (45.5)
```

5. ROOT Essentials (The "Must Haves")

A. Random Numbers (The "MC" in MC Lectures)



Used to generate simulated data.

```
TRandom3 *r = new TRandom3(0); // 0 = Random seed based on time
double x = r->Uniform(0, 10);  // Random flat number 0 to 10
double y = r->Gaus(90, 5);     // Gaussian (Mean 90, Sigma 5)
```

B. Histograms (TH1F) Used to store the result.

```
// (InternalName, Title, Bins, Min, Max)
TH1F *h = new TH1F("h1", "My Plot", 100, 0, 100);

h->Fill(45.2); // Add data
h->Draw();     // Visualize it
```

C. Physics Vectors (`TLorentzVector`) Used to handle relativity logic automatically.

```
TLorentzVector mu;
mu.SetPtEtaPhiM(30.0, 2.1, 1.5, 0.105); // Define vector

double E = mu.E(); // Get Energy
double M = mu.M(); // Get Mass
```

6. Math Cheat Sheet

ROOT uses the `TMath` library for complex math.

- **Square Root:** `TMath::Sqrt(x)` or `sqrt(x)`
- **Absolute Value:** `TMath::Abs(x)` or `abs(x)`
- **Power:** `TMath::Power(x, 2)` or `pow(x, 2)`
- **Pi:** `TMath::Pi()`

7. File I/O (Skeleton Code)

Every analysis script usually looks like this.

```
void Analyze() {
    // 1. Open File
    TFile *file = new TFile("data.root", "READ");

    // 2. Get the Tree (Table of data)
    TTree *tree = (TTree*)file->Get("Events");

    // 3. Link variables (boilerplate usually goes here)
    // ...

    // 4. Loop
    int n = tree->GetEntries();
    for(int i=0; i<n; i++) {
        tree->GetEntry(i);
        // Do Analysis...
    }
}
```