Laboratorio di Elettromagnetismo e Ottica (A.A. 2023-2024)
Parte di Programmazione C++/ROOT
Seconda Prova

Punto 7)

- aggiungere alla classe Particle i seguenti metodi (la cui implementazione –file extra.cpp -potete scaricare daVirtuale, adattandolo -se necessario- opportunamente ai nomi che avete scelto per i metodi/attributi delle classi Particle/ParticleType utilizzati nel vostro codice):
  - -Un metodo pubblico che simula attraverso un generatore random la cinematica del decadimento di una particella in altre due particelle:
    int Decay2Body(Particle &dau1,Particle &dau2);
    uso: p1.Decay2body(p2,p3); //Fa decadere p1 in p2 e p3. N.B p1 e p2 sono istanze e non puntatori.
    Dopo la chiamata p2 e p3 contengono gli impulsi finali delle figlie del decadimento. Il risultato (int) ritorna 0 se tutto è andato a buon fine, un numero maggiore di 0 se c'è stato un errore
  - Un metodo privato che serve al metodo Decay2Body per eseguire il decadimento (trasformazione di Lorentz),
    void Boost(double bx, double by, double bz);
- affinche' il metodo Decay2Body possa funzionare in modo trasparente (nel suo codice si utilizza l'array nativo (o array/vector stl, a seconda della vostra implementazione) di puntatori a ParticleType (classe base di ResonanceType), se non già fatto, occorre implementare anche nella classe ParticleType il metodo che ritorna la larghezza (il GetWidth()), rendendolo virtuale e facendogli ritornare 0 (una ParticleType è considerata "particella stabile" ai fini pratici).

Punto 8)

Compilate le classi ParticleType, ResonanceType e Particle all'interno di ROOT, utilizzando uno dei comandi illustrati a Lezione. Verranno prodotte delle librerie dinamiche (.so), per ciascuna classe, che saranno a supporto del main program. Se eseguite il programma da prompt interattivo di ROOT, occorre caricare queste librerie (per esempio con: .L ParticleType.cxx+, .L ResonanceType.cxx+, .L Particle.cxx+) prima della compilazione/esecuzione del "main module", che gestirà la generazione Monte Carlo.

Punto 9)

Scrivere un "main module", che compilerete all'interno di ROOT come le tre classi di supporto:

0) Come prima operazione inserire attraverso il metodo AddParticleType i tipi di particelle che verranno successivamente generati (i valori di massa, carica e larghezza sono riportati nel punto 8.3. Le cariche sono espresse in unità di carica (valore assoluto) dell'elettrone, e le masse/larghezze sono in unità GeV/c^2). In questa fase fate anche gRandom→SetSeed() per inizializzare il seed della generazione.

1) Generare 10^5 eventi: ogni evento è costituito da 100 particelle (sugg: per gestirle, una opzione particolarmente efficiente è usare un array nativo di istanze di tipo "Particle EventParticles[N]" (ma potete utilizzare anche un array stl/vector stl) che va definito all'esterno dei cicli (perché?), prima di iniziare la generazione, e che sovrascriverete a ogni evento dopo aver salvato le informazioni necessarie negli istogrammi di cui al punto 9.5). Poiché in media in ogni evento produrrete una risonanza K*, che decade in altre due particelle (si veda punto 9.4) la dimensione/size "safe" del contenitore "EventParticles" di Particle scelto è tipicamente maggiore di 100 (per esempio 120 è una scelta adeguata).

2) Per ogni particella, generare:
· secondo una distribuzione uniforme fra $[0,2\pi]$ la coordinata azimutale phi
· secondo una distribuzione uniforme fra $[0,\pi]$ la coordinata polare theta
· l'impulso della particella secondo una distribuzione esponenziale decrescente con <p> =1 GeV
Utilizzate i metodi di generazione Monte Carlo di ROOT (TRandom::Rndm(), TRandom::Uniform(),TRandom::Exp(double mean)).
Potete poi "settare" Px,Py,Pz (applicare trasformazione coordinate polari→coordinate cartesiane)

dell'istanza corrente di tipo Particle (nata con il costruttore di default) utilizzando il metodo SetP(...)che avete implementato nella prima prova.

3) In ciascun evento generate random i seguenti tipi di particelle secondo le proporzioni (positive e negative in egual proporzione, per esempio 40% pioni + e 40% pioni-, etc.):
pioni (+/-) 80% (m_π=0.13957 GeV/c^2, q_π=+/-1)
Kaoni(+/-10% (m_K=0.49367 GeV/c^2, q_K=+/-1)
protoni (+/-) 9% (m_p=0.93827 GeV/c^2, q_p=+/-1)
K* (risonanza) 1% (m_{k*}=0.89166 GeV/c^2, q_{k*}=0, Γ_{k*}=0.050 GeV/c^2,)
Potete "settare" il tipo di particella (proprietà di base) dell'istanza corrente di tipo Particle (nata con il costruttore di default) utilizzando il metodo SetIndex(...)che avete implementato nella prima prova.

4) Nel caso che la particella generata sia un K*, fare decadere la risonanza utilizzando il metodo "Particle::Decay2body(…)" (50% pione+ e Kaone -, 50% pione- e Kaone+)
N.B. Dopo il decadimento il numero di particelle generate potrà quindi essere maggiore di 100.
Suggerimento: aggiungete le figlie della K* in coda al contenitore EventParticles, dopo il centesimo elemento.

5) Definite (all'inizio, fuori dai cicli, e chiamando contestualmente anche il metodo Sumw2() negli istogrammi di massa invariante per una corretta valutazione delle incertezze sui contenuti dei bin nelle operazioni -occorrerà fare delle differenze- di questi istogrammi)) e riempite (all'interno dei cicli) utilizzando le classi e relativi metodi del pacchetto ROOT i seguenti istogrammi delle proprietà dalle particelle generate:
· tipi di particelle generati
· distribuzioni degli angoli azimutali e polari
· distribuzione dell'impulso, dell'impulso trasverso sqrt(p_x^2+p_y^2) e dell'energia della particella
· massa invariante fra tutte le particelle generate in ciascun evento, evento per evento (per la massa invariante, utilizzare il metodo Particle::InvMass(…))
· Per scelta di range, numero di bin dei vari istogrammi ci confrontiamo durante il turno (ma vi invito a fare autonomamente le vostre considerazioni, prima, come "esercizio" e spunto di discussione).

· massa invariante fra tutte le particelle generate in ogni evento, in combinazioni con carica di segno discorde
· massa invariante fra tutte le particelle generate in ogni evento, in combinazioni con carica di segno concorde
· massa invariante fra tutte le particelle generate in ogni evento con combinazioni di tipo pione+/Kaone- e pione-/Kaone+
-massa invariante fra tutte le particelle generate in ogni evento con combinazioni di tipo pione+/Kaone+ e pione-/Kaone-
· massa invariante fra le particelle generate in ogni evento che derivano dal decadimento della risonanza K* (quelle in "coda", per intenderci. N.B. considerate esclusivamente coppie di particelle figlie che provengono dalla stessa "madre", i.e. non mischiare "figlie" di "madri" diverse, poiché solo nel primo caso si osserverà in questo istogramma il caratteristico picco di massa invariante). Questo istogramma fa da istogramma di benchmark affinché si possa essere certi che la gestione del decadimento, le formule di massa invariante/energia e la virtualizzazione delle classi ParticleType/ResonanceType siano state correttamente implementate. Se tutto è ok in questa fase, dovreste osservare **una gaussiana con media la massa della K*, e RMS la larghezza della K* come impostata all'inizio del programma**.
6) Salvare tutti gli istogrammi su un unico file ROOT, per una successiva analisi. L'analisi sarà gestita da un altro programma/macro, che prenderà questo file come input, e sarà tema della terza prova.