Laboratorio di Elettromagnetismo e Ottica (A.A. 2021-2022)
Parte di Programmazione C++/ROOT
S. Arcelli
Seconda Prova

Punto 7)

- aggiungere alla classe Particle i seguenti metodi (la cui implementazione –extra.cpp-potete scaricare da Insegnamenti Online, adattandolo -se necessario- opportunamente ai nomi che avete scelto per i metodi/attributi della classe Particle utilizzati nel vostro codice):
  - -Un metodo pubblico che simula attraverso un generatore random la cinematica del decadimento di una particella in altre due:
    int Decay2Body(Particle &dau1,Particle &dau2);
    uso: p1.Decay2body(p2,p3); //Fa decadere p1 in p2 e p3.
    Dopo la chiamata p2 e p3 contengono gli impulsi finali del decadimento. Il risultato (int) ritorna 0 se tutto e' andato a buon fine, un numero maggiore di 0 se c'e' stato un errore
  - Un metodo privato che serve al metodo Decay2Body per eseguire il decadimento, **void Boost(double bx, double by, double bz);**
- affinche' il metodo Decay2Body possa funzionare in modo trasparente utilizzando i puntatori a ParticleType (classe base di ResonanceType),se non già fatto, occorre implementare anche nella classe ParticleType il metodo che ritorna la larghezza (GetWidth()), rendendolo virtuale e facendogli ritornare 0.

Punto 8)

Nel main module:

0) 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).

1) Generare 10^5 eventi: ogni evento e' costituito da 100 particelle (sugg: per descriverle una opzione particolarmente efficiente è usare un array di tipo "Particle particella[N]; con N dimensionato per esempio a 120), che va definito all'esterno dei cicli, prima di iniziare la generazione, e che sovrascriverete a ogni evento dopo aver salvato le informazioni necessarie negli istogrammi di cui al punto 8.5. Poichè in media in ogni evento produrrete una risonanza K*, che decade in altre due particelle-si veda punto 8.4, dimensionare l'array a N maggiore di 100 (per esempio 120).

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 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, □=0.050 GeV/c^2)
Potete "settare" il tipo 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 potra' quindi essere maggiore di 100.
Suggerimento: aggiungerere le figlie della K* in coda all'array, dopo il centesimo elemento.

4) Definite e riempite (utilizzando le classi e relativi metodi del pacchetto ROOT) i seguenti istogrammi delle proprieta' dalle particelle generate:
· tipi di particelle generati
· distribuzioni degli angoli azimutali e polari
· distribuzione dell'impulso, dell'impulso trasverso sqrt(px^2+py^2) e dell'energia della particella
· massa invariante fra tutte le particelle generate in ogni evento (per la massa invariante, utilizzare il metodo Particle::InvMass)

· 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* (N.B. considerate solo coppie di particelle figlie che provengono dalla stessa "madre")

5)Salvare gli istogrammi su un file di ROOT, per una successiva analisi.