Laboratorio di Elettromagnetismo e Ottica– C++/ROOT
Lezione VII

Silvia Arcelli

Laboratorio di programmazione C++/ROOT

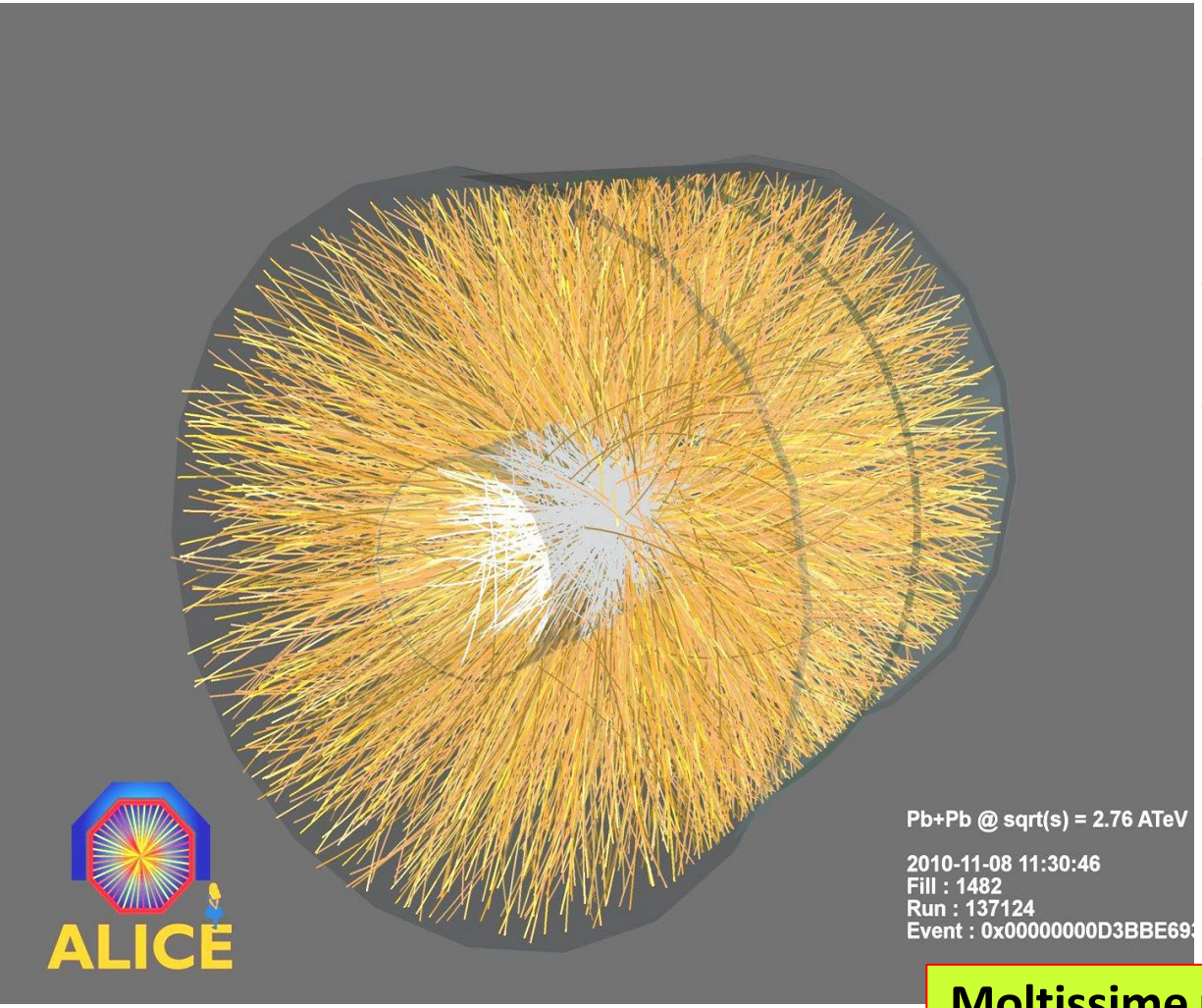
- **Scopo della prova: implementare un prototipo di codice utilizzabile per rappresentare e analizzare il contenuto di eventi fisici simulati risultanti da collisioni di particelle elementari**

  - **Fare pratica sulla programmazione a oggetti: meccanismi di ereditarietà e aggregazione, utilizzo di attributi/metodi statici e const, polimorfismo al run-time (ereditarietà virtuale)**

  - **Fare pratica di generazione Monte Carlo con ROOT: utilizzeremo la generazione secondo definite proporzioni, secondo la distribuzione esponenziale, la distribuzione uniforme, la distribuzione gaussiana.**

  - **Fare pratica di analisi dati con ROOT: verifica della correttezza delle distribuzioni generate, ed estrazione di un segnale di una particella instabile con vita media molto breve (la K*) attraverso l'analisi della distribuzione in massa invariante dei suoi prodotti di decadimento.**

Laboratorio di programmazione C++/ROOT



- N particelle/evento~10^{2}-10^{4}
- N eventi~10^{5}-10^{7}
- Tipi di particelle:
  e, π, K, μ, p, n, γ…
  + risonanze (come la K*).

Moltissime particelle in ciascun evento, ma di un numero limitato di tipi

Laboratorio di programmazione C++/ROOT

Abbiamo quindi questo problema:

- **Generare molti eventi (10^5)**
- **In ciascun evento ci sono molte particelle (~100), ma il loro tipo è limitato. Genereremo:**

- **40% π^+**
- **40% π^+**
- **5% K^+**
- **5% K^-**
- **4.5% P^+**
- **4.5% P^-**
- **1% K^{0*}**

Proprietà "di base" delle particelle:
- **nome,**
- **massa,**
- **carica,**
- **eventualmente un altro parametro, la larghezza Γ, legata alla vita media della particella.**

due classi per rappresentare le proprietà di base delle particelle: ParticleType e ResonanceType

Classi ParticleType e ResonanceType

ParticleType è una classe che descrive tre proprietà di base di una particella stabile, includendoli come attributi (di tipo const):

Nome, Massa ,Carica

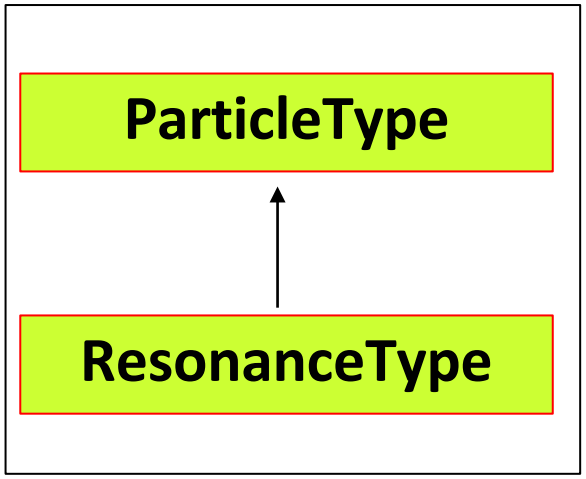
ResonanceType è una classe che descrive le quattro proprietà di base di una particella instabile (risonanza), sempre includendoli come attributi (sempre di tipo const):

Nome, Massa ,Carica, Larghezza

ResonanceType è in sostanza una ParticleType con un attributo in più, la Larghezza di risonanza.

Classi ParticleType e ResonanceType

Essendo ResonanceType chiaramente un tipo specializzato di ParticleType (relazione "is a"), una scelta ovvia per riutilizzare il codice scritto per ParticleType è far ereditare ResonanceType da ParticleType (si veda lezione 3 per riferimento)**.**



Dovremo aggiungere un attributo, implementare un metodo specializzato, ridefinire qualche metodo. Tutti gli attributi di ParticleType e ResonanceType sono di tipo const, occorre inizializzarli opportunamente (cioè attraverso la lista di inizializzazione, si veda lezione 2).

Classe Particle

A queste particelle poi dobbiamo conferire un impulso P (3-D) che può variare da particella a particella, da evento a evento .

$$\vec{P} = (P_x, P_y, P_z)$$

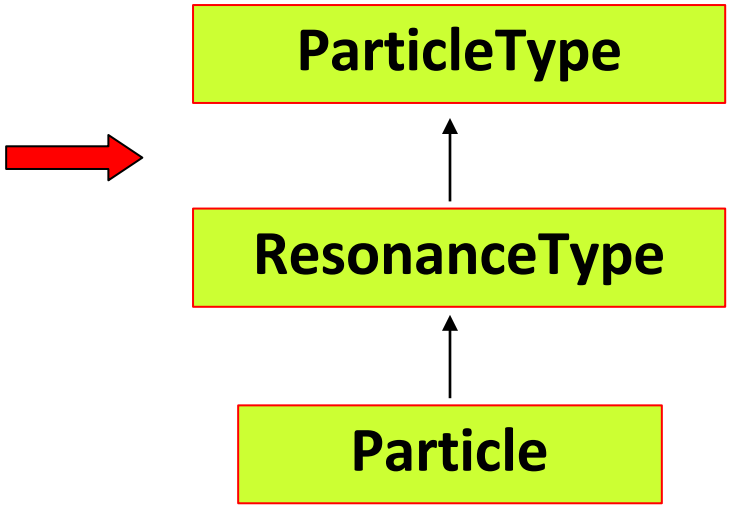
Dobbiamo quindi scrivere una classe ulteriormente specializzata, che contenga sia le informazioni delle proprietà di base (nome, massa, carica, eventualmente larghezza di risonanza), sia le proprietà cinematiche (impulso 3-D):

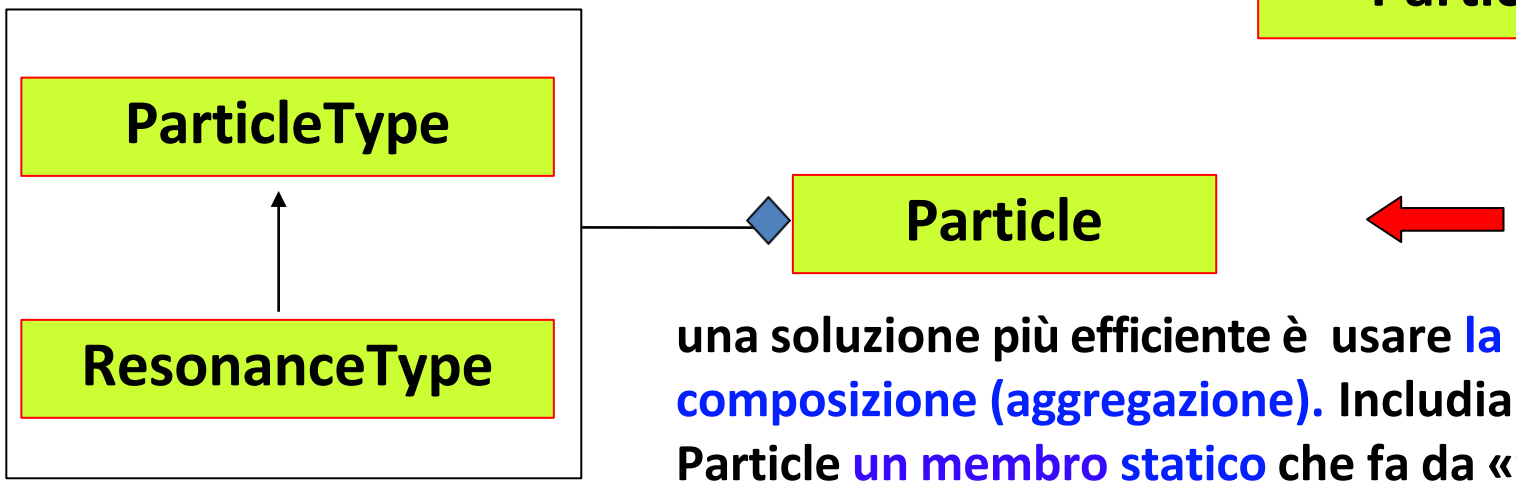
Classe **Particle**

possiamo fare due scelte: un classe Particle che eredita da ResonanceType, o una strada diversa: l'aggregazione (composizione) (si veda lezione 3 per riferimento)

Classe Particle

Se scegliamo di utilizzare esclusivamente l'ereditarietà, " funziona", ma nell'insieme di oggetti Particle che il programma istanzia, le stesse proprietà di base (massa,carica,nome) sarebbero duplicate inutilmente per tantissimi oggetti.





una soluzione più efficiente è usare la composizione (aggregazione). Includiamo in Particle un membro statico che fa da «tabella» per i tipi di particella e relative proprietà di base: Risparmio di memoria!

Classe Particle-Attributi

Quali sono gli attributi di un oggetto di tipo Particle?

- **le tre componenti dell'impulso(Px,Py,Pz)**
- **un indice intero (fIndex)**

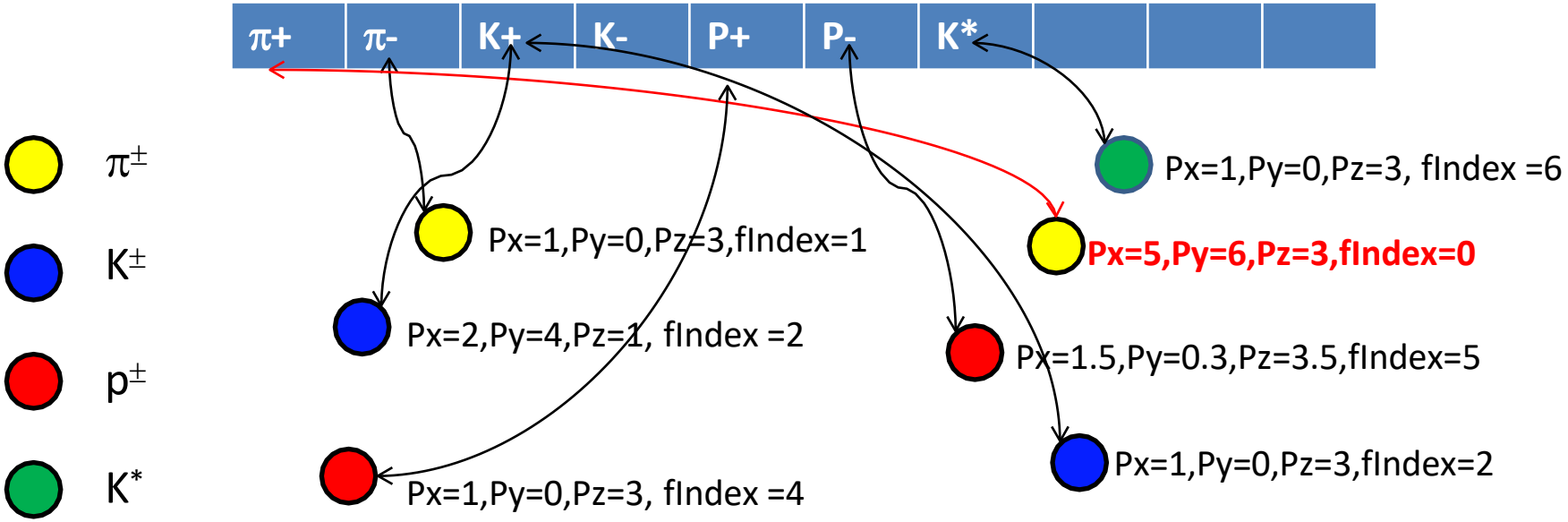
Scritti per ogni istanza

- **Un membro statico che è un array di puntatori a ParticleType (puntatori, e non istanze, per utilizzare appieno ereditarietà virtuale e polimorfismo al run-time). Ha la funzione di "tabella" per descrivere le proprietà di base del tipo di particella di un oggetto Particle (nome, massa,carica, eventualmente larghezza) .**

Comune a tutte le istanze, non duplicato! →Risparmio di memoria

Classe Particle-Attributo fIndex

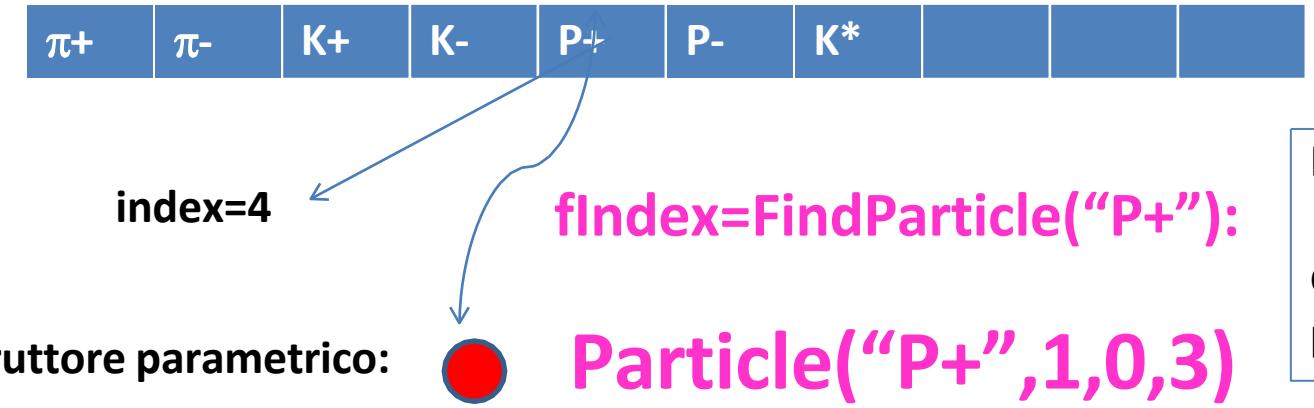
Ciascuna istanza di Particle, per esempio un pione positivo con componenti dell'impulso(Px=5,Py=6,Pz=3), recupera le sue informazioni di base (nome, massa, carica ed eventualmente Larghezza) attraverso un indice intero (fIndex), che corrisponde alla posizione del tipo «pione positivo» nella "tabella" (ossia l'i-esimo elemento dall'array statico di puntatori a oggetti di tipo di base ParticleType)



Il metodo FindParticle

Quindi a ciascuna istanza di Particle è univocamente associata l'informazione delle proprietà di base attraverso l'attributo fIndex, che nel codice (nel costruttore di Particle) faremo determinare attraverso una chiamata al metodo (privato e statico) int FindParticle(const char*name).

Questo metodo, in base al nome fornito dall'utente nel creare l'istanza di Particle (quindi nella lista di argomenti del costruttore di Particle), deve cercare nella "tabella" di puntatori statici l'indice dell'elemento con nome corrispondente al nome fornito nel costruttore. Tale indice deve poi essere utilizzato per assegnare il valore corretto al membro fIndex della classe Particle



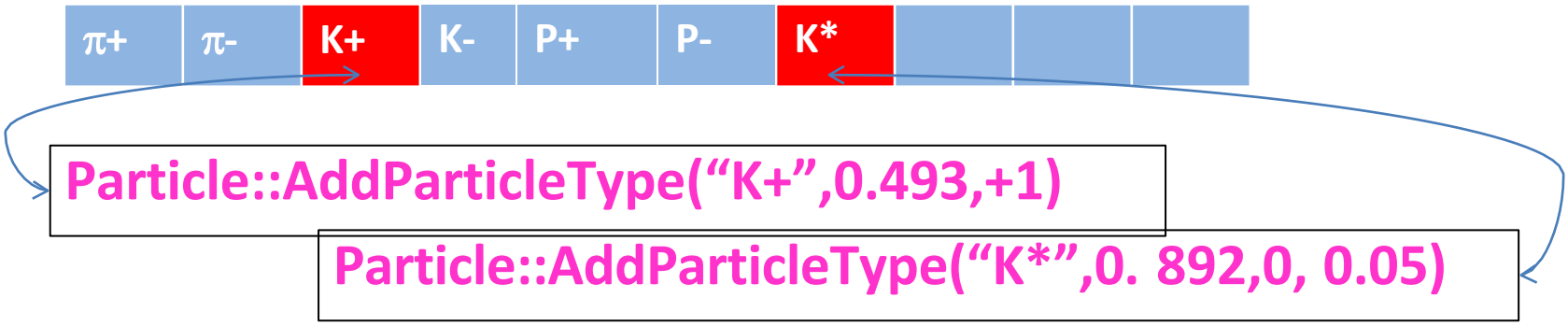
Nel corpo (o nella lista di inizializzazione) del costruttore parametrico

Il metodo AddParticleType

- **L'array statico di puntatori a ParticleType è inizialmente vuoto, a esso non è associata alcuna memoria. In Particle implementeremo un metodo statico (AddParticleType)**

void Particle::AddParticleType(name,mass,charge,width)

che ha la funzione di riempirlo sequenzialmente (attraverso l'utilizzo dell'allocazione dinamica): poiché è un metodo statico, può essere usato indipendentemente dall'aver creato istanze di Particle (e quindi proprio all'inizio della funzione principale di simulazione)



Il decadimento della K*

Nel caso in cui la Particle sia una particella instabile (K*), a essa va riservato un trattamento un po' particolare: occorre farla decadere.

$$K^* \to \pi^+ K^- \quad 50\%$$
$$K^* \to \pi^- K^+ \quad 50\%$$

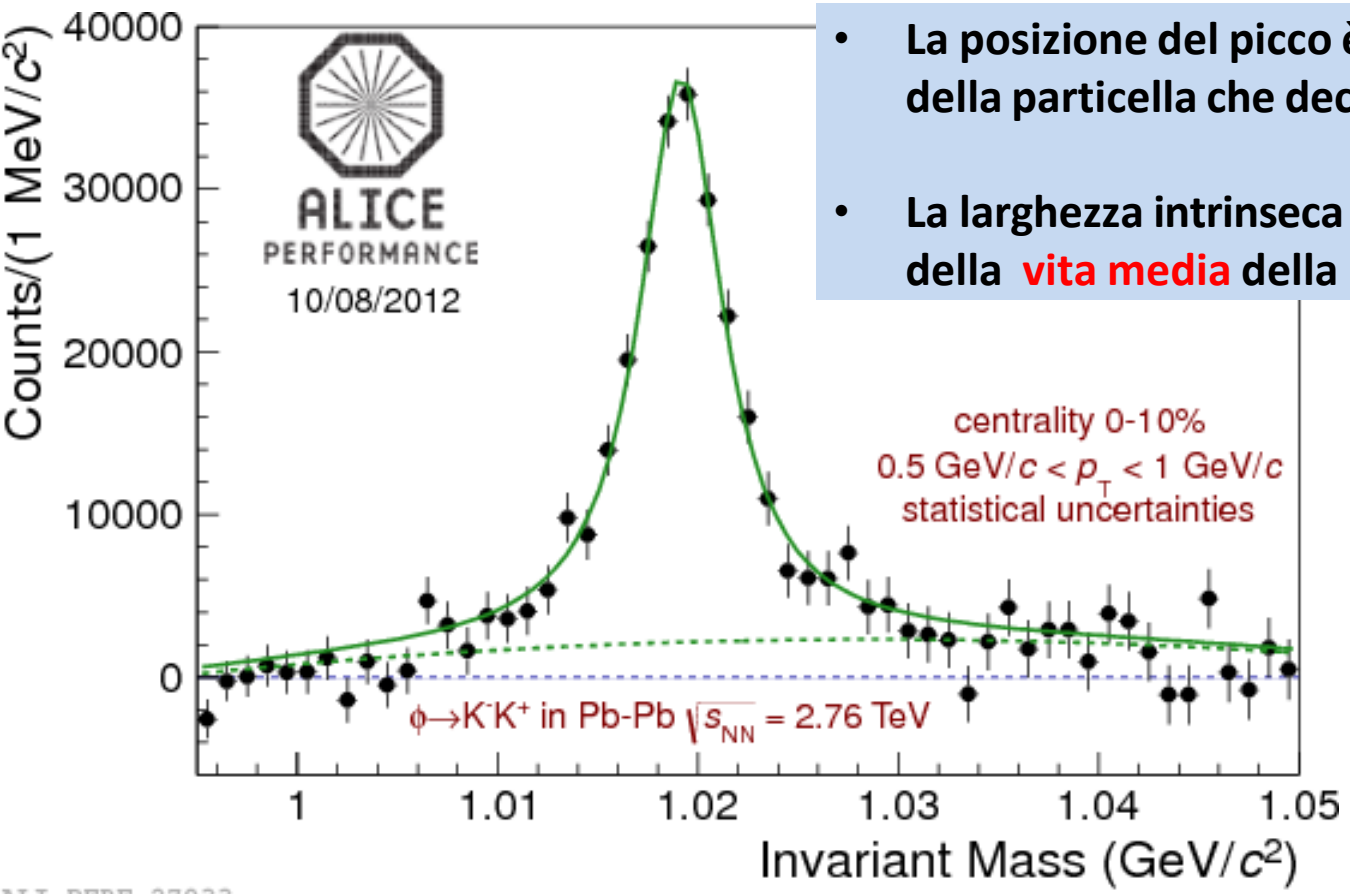
Metodo Particle:.Decay2Body(...)
(già fornito, non occorre implementarlo)

E ricostruire dai suoi prodotti di decadimento πK la Massa Invariante:

$$M_{12} = \sqrt{(E_1 + E_2)^2 - \left|\vec{P}_1 + \vec{P}_2\right|^2} \qquad E = \sqrt{m^2 + \left|\vec{P}\right|^2}$$

M_{12} è un invariante relativistico

Segnale nella distribuzione di Massa invariante



- **La posizione del picco è legata alla massa della particella che decade**
- **La larghezza intrinseca è legata all'inverso della vita media della particella**

ALI-PERF-27033

Ia Prova

SCRIVERE LE CLASSI NECESSARIE PER IL PROGRAMMA

Tre classi di supporto al programma:

- **ParticleType**
- **ResonanceType**

descrittive delle proprietà di base - nome, massa, carica, (e anche larghezza, per ResonanceType)

- **Particle**

Aggiunge proprietà cinematiche: **non eredita** da ParticleType/ResonanceType, ma **include** degli oggetti di tal tipo attraverso un array di puntatori a ParticleType (composizione, non ereditarietà)

IMPORTANTE

Separare dichiarazione di classe e sua implementazione in files di tipo .h e .cxx, ricordare di mettere include guard

Per quanto riguarda i metodi:

- **Utilizzate ereditarietà virtuale**
- **dichiarate const i metodi dichiarabili const**

Turni di Laboratorio: Informazioni Pratiche

-per ogni prova scaricate le traccia disponibile in anticipo su Virtuale, come il codice aggiuntivo (necessario per la II prova)

-per le primo turno non è strettamente necessario avere una installazione di ROOT funzionante, è sufficiente avere il compilatore C++

-per utilizzare i PC di laboratorio:

- ✓ **accedere con le proprie credenziali istituzionali**

IIª Prova

GENERAZIONE MONTE CARLO

Implementare metodi aggiuntivi della classe Particle, e scrivere il programma di generazione Monte Carlo.

Prima di scrivere il programma di generazione:

- **fate un test del codice scritto precedentemente, inserendo gli elementi richiesti dalla traccia nell'array statico di puntatori a Particletype (membro della classe Particle), attraverso il metodo AddParticleType(...)**
- **Stampate il contenuto dell'array statico utilizzando il metodo (PrintArray()) che vi è stato richiesto di implementare**
- **Istanziate delle Particle per controllare se i metodi che avete scritto fanno quello che vi aspettate....**

IIª Prova

Generazione "Monte Carlo" di eventi fisici : 10^5 eventi, ognuno contenente 100 particelle di alcuni tipi predefiniti, fra cui uno stato risonante che può decadere (K*).

Generare:

1) Distribuzione uniforme nelle direzioni (angoli θ e ϕ)

2)Distribuzione esponenziale del modulo dell'impulso

3) Secondo definite proporzioni (date dalla traccia) dei tipi di particelle

4) Nel caso in cui sia generata una K*, farla decadere in una coppia πK di carica opposta con proporzione 50%,50%.

IIª Prova

Schema suggerito:

1) Riempire l'array di puntatori a ParticleType attraverso il metodo statico AddParticleType, prima della creazione di ogni istanza

2) Definire un array nativo statico (non dinamico) del tipo:
Particle particles[100+x] (x dell'ordine di 20-30, per contenere eventuali figlie del decadimento della risonanza). Alternativamente, potete utilizzare anche un stl array o un vector

questo array funge da contenitore provvisorio e sarà sovrascritto a ogni evento (occorrerà anche implementare il costruttore di default di Particle).

3) Impostare un doppio ciclo (2 for annidati), quello esterno su 10^5 eventi e quello interno su 100 particelle

IIª Prova

Per ogni evento, generate 100 particelle:

- **Proprietà cinematiche:Px,Py,Pz**
  - **Distribuzione uniforme nelle direzioni (angoli θ e ϕ)**
  - **Distribuzione esponenziale del modulo dell'impulso**

Utilizzare il metodo Particle::SetP(Px,Py,Pz) per assegnarle all'istanza

- **Secondo definite proporzioni (date dalla traccia) dei tipi di particelle**
  - **Impostare blocco if –else if-…- else**

Utilizzare il Setter dell'attributo fIndex Particle::SetIndex(nome/indice) per assegnare l'identità corretta all'istanza

- **Nel caso in cui sia generata una K*, farla decadere in πK di carica opposta con proporzione 50%,50%. Utilizzare il metodo Particle::Decay2Body(...) per il decadimento.**

Inserire le figlie del decadimento nell'array particles[], a partire dalla posizione successiva alla centesima dell'array (le figlie «in coda»).

IIª Prova

Durante la generazione **riempire gli istogrammi** delle proprietà delle particelle, ovvero **modulo dell'impulso, angolo polare e azimutale, energia.** **Per queste distribuzioni, fate riferimento solo alle particelle «primarie» (le prime 100), escludendo le «figlie» dei decadimenti.**

Riempire gli istogrammi di massa invariante (e qui bisogna includere anche le «figlie»!). In ogni evento, fare le combinazioni a due a due:

- **Massa invariante fra tutte le particelle di carica concorde**
- **Massa invariante fra tutte le particelle di carica discorde**
- **Massa invariante fra tutte le combinazioni π,K di carica concorde**
- **Massa invariante fra tutte le combinazioni π,K di carica discorde**

- **Istogramma di controllo: Massa invariante fra i prodotti di decadimento della K* "veri" (da riempire nella parte di codice in cui generate una K*, solo combinazioni dalla stessa K*).**

IIª Prova

La K* è un segnale relativamente raro, e nella distribuzione di massa invariante delle combinazioni a due a due il suo "picco" di risonanza è sommerso dal fondo di combinazioni accidentali. Tuttavia, utilizzando l'informazione della carica opposta delle particelle e successivamente anche la loro identità (tipo), si riesce a osservare il segnale **per sottrazione**.

La logica è :

1) Massa invariante fra tutte le particelle di carica concorde

→ solo combinazioni accidentali, non possono venire dalla K*

1) Massa invariante fra tutte le particelle di carica discorde

2)→ combinazioni accidentali +K*

2)-1) = K*

Con selezione dei tipi di particella (π,K), ancora più efficace.

IIIª Prova

ANALISI

Analisi degli eventi generati attraverso gli istogrammi salvati su file root alla fine del ciclo di generazione attraverso una macro indipendente, che legge gli istogrammi salvati sul file root e li analizza:

- **Verificare che le distribuzioni di impulso e angoli polari e azimutali siano coerenti con quanto simulato in fase di generazione attraverso un fit (deve dare un buon X2/NDF quando adattate alle distribuzioni aspettate, expo per impulso e pol0 per angoli).**

- **Dalla distribuzione di massa invariante, utilizzando i metodi di sottrazione degli istogrammi, separare il segnale della risonanza K* dal fondo di altre particelle, seguendo le indicazioni date dalla traccia (sottrazione istogrammi).**

- **Attraverso un fit del segnale della K* (assumere una distribuzione gaussiana), estrarre i parametri della risonanza (media = massa, sigma=larghezza) con relative incertezze e confrontarli con quelli impostati in fase di generazione.**

Relazione-Struttura Generale

Struttura della relazione:

1) **Introduzione: descrivere brevemente lo scopo del programma**

2) **Struttura del codice: quali classi sono state implementate, con che funzione, quali meccanismi di reimpiego di codice sono stati usati, e perchè**

3) **Generazione: quanti eventi sono stati generati, quante particelle, quali tipi di particelle e con che proporzioni, come sono state generate le proprietà cinematiche.**

4) **Analisi: Discutere la congruenza delle distribuzioni osservate con i dati in input alla generazione, spiegare brevemente l'approccio seguito per estrarre il segnale della risonanza**

Per ciascuna sezione max 250-300 parole

Relazione-Sezione Analisi

Presentare i risultati **in una tabella** –**Abbondanze delle particelle:**

specie	occorrenze osservate	occorrenze attese
π+	(X±dX)	X
π-	(X±dX)	X
K+	(X±dX)	X
K-	(X±dX)	X
p+	(X±dX)	X
P-	(X±dX)	X
K*	(X±dX)	X

Relazione-Sezione Analisi

Presentare i risultati **in una tabella** - Distribuzione **angoli polari e azimutali, modulo dell'impulso:**

distribuzione	Parametri del Fit	χ^2	DOF	χ^2/DOF
Fit a distribuzione angolo θ (pol0)	(X$\pm$dX) Il parametro 0	X	X	X
Fit a distribuzione angolo ϕ (pol0)	(X$\pm$dX) Il parametro 0	X	X	X
Fit a distribuzione modulo impulso (expo)	(X$\pm$dX) Media EXP	X	X	X

Relazione-Sezione Analisi

Presentare i risultati **in una tabella** - **Analisi della K***

Distribuzione e fit	Media	Sigma	Ampiezza	χ^2/DOF
Massa Invariante vere K* (fit gauss)	(X±dX)	(X±dX)	(X±dX)	X
Massa Invariante ottenuta da differenza delle combinazioni di carica discorde e concorde (fit gauss)	(X±dX)	(X±dX)	(X±dX)	X
Massa Invariante ottenuta da differenza delle combinazioni πK di carica discorde e concorde (fit gauss)	(X±dX)	(X±dX)	(X±dX)	X

Relazione-Sezione Analisi

Inserire le **seguenti figure**:

• distribuzioni di **abbondanza di particelle, dell'impulso, angolo polare, angolo azimutale** (unica Canvas divisa in 4) con fit sovrapposti e parametri stampati nella box della statistica. Mettere i titoli agli assi.

• **Massa invariante** delle K* vere, massa invariante ottenuta dalla differenza fra combinazioni di carica discorde e concorde, massa invariante ottenuta dalla differenza fra combinazioni πK di carica discorde e concorde (unica Canvas divisa in 3), con fit sovrapposti e parametri stampati nella box della statistica. Mettere i titoli agli assi.

Relazione-Appendice

Importante:

In appendice allegare un listato stampato del codice (Classi e programma di generazione e analisi), sempre in formato pdf e all'interno del file che contiene la relazione (no file separati). Diversamente la relazione sarà considerata non valutabile.

Relazione-Consegna

Scadenza di consegna della relazione: 10/01/2024

- **Strettamente in formato pdf**
- **Via e-mail a silvia.arcelli@unibo.it, nicolò.jacazio2@unibo.it con copia al tutor elisa.sanzani@studio.unibo.it, da un indirizzo di posta istituzionale (mi raccomando!), e recante come subject del mail, in maiuscolo,**

COGNOME_NOME_RELAZIONE_ROOT_2024

dove cognome e nome corrispondono a quelli del primo firmatario (ordine alfabetico). utilizzate a stessa convenzione per denominare il file pdf allegato. Punteggio: max 5 punti

Se non si consegna la relazione entro questa data, il punteggio assegnato alla relazione sarà 0 punti.