generator.cpp

```cpp
#include "particle.hpp"

#include <TApplication.h>
#include <TCanvas.h>
#include <TFile.h>
#include <TH1D.h>
#include <TROOT.h>
#include <TRandom3.h>
#include <TSystem.h>

#include <iostream>

void randomParticlePosition(TRandom3* rand, Particle& particle);

Particle createRandomParticle(TRandom3* rand);

void fillHistogram(const std::array<Particle, 120>& particle, TH1D* hInvariantMass,
                   TH1D* hMassOppositeSign, TH1D* hMassSameSign,
                   TH1D* hMassPionKaonOpposite, TH1D* hMassPionKaonSame,
                   TH1D* hType, TH1D* hEnergy, TH1D* hTheta, TH1D* hPhi,
                   TH1D* hPout, TH1D* hPtrasv, int n_particles);

void createInstances(int n_event, int n_particles_event, TRandom3* rand,
                     std::array<Particle, 120>& event_particles,
                     TH1D* hMassKStarDecay, TH1D* hInvariantMass,
                     TH1D* hMassOppositeSign, TH1D* hMassSameSign,
                     TH1D* hMassPionKaonOpposite, TH1D* hMassPionKaonSame,
                     TH1D* hType, TH1D* hEnergy, TH1D* hTheta, TH1D* hPhi,
                     TH1D* hPout, TH1D* hPtrasv);

void saveHistograms(const std::array<TH1D*, 12>& histograms,
                    const std::string&           filename);

int main() {
  // Crea un'applicazione ROOT
  TApplication theApp("App", nullptr, nullptr);

  std::cout << "Generating canvas..." << std::endl;
  TCanvas* canvas = new TCanvas("c1", "Generation", 800, 600);
  canvas->Divide(4, 3); // Dividi il canvas in 3 righe e 4 colonne

  // name, mass, charge, whidth
  Particle::addParticleType('Q', 0.13957, 1);
  Particle::addParticleType('q', 0.13957, -1);
  Particle::addParticleType('P', 0.93827, 1);
  Particle::addParticleType('p', 0.93827, -1);
  Particle::addParticleType('K', 0.49367, 1);
  Particle::addParticleType('k', 0.49367, -1);
```

```
49    Particle::addParticleType('*', 0.89166, 0, 0.050);
50
51    int                       n_event           = 100000;
52    int                       n_particles_event = 100;
53    std::array<Particle, 120> event_particles;
54
55    // Istogrammi
56    TH1D* hInvariantMass =
57        new TH1D("hInvariantMass", "Invariant Mass distribution", 600, 0, 6);
58    hInvariantMass->Sumw2();
59    TH1D* hMassOppositeSign = new TH1D(
60        "hMassOppositeSign", "Invariant Mass - Discordant Charge", 600, 0, 6);
61    hMassOppositeSign->Sumw2();
62    TH1D* hMassSameSign = new TH1D(
63        "hMassSameSign", "Invariant Mass - Concordant Charge", 600, 0, 6);
64    hMassSameSign->Sumw2();
65    TH1D* hMassPionKaonOpposite = new TH1D(
66        "hMassPionKaonOpposite", "Invariant Mass- Pi/K discordant", 600, 0, 6);
67    hMassPionKaonOpposite->Sumw2();
68    TH1D* hMassPionKaonSame = new TH1D(
69        "hMassPionKaonSame", "Invariant Mass - Pi/K concordant", 600, 0, 6);
70    hMassPionKaonSame->Sumw2();
71    TH1D* hMassKStarDecay =
72        new TH1D("hMassKStarDecay", "Invariant Mass - K* decay", 40, 0.7, 1.1);
73    hMassKStarDecay->Sumw2();
74
75    TH1D* hType = new TH1D("hType", "Particle type distribution", 7, 0, 7);
76    hType->SetStats(0);
77    hType->GetXaxis()->SetBinLabel(1, "#pi+");
78    hType->GetXaxis()->SetBinLabel(2, "#pi-");
79    hType->GetXaxis()->SetBinLabel(3, "P+");
80    hType->GetXaxis()->SetBinLabel(4, "P-");
81    hType->GetXaxis()->SetBinLabel(5, "K+");
82    hType->GetXaxis()->SetBinLabel(6, "K-");
83    hType->GetXaxis()->SetBinLabel(7, "K*");
84
85    TH1D* hEnergy = new TH1D("hEnergy", "Energy distribution", 600, 0, 6);
86    TH1D* hTheta  = new TH1D("hTheta", "Theta distribution", 314, 0, M_PI);
87    TH1D* hPhi    = new TH1D("hPhi", "Phi distribution", 628, 0, 2 * M_PI);
88    TH1D* hPout   = new TH1D("hPout", "Momentum distribution", 600, 0, 6);
89    TH1D* hPtrasv =
90        new TH1D("hPtrasv", "Transverse momentum distribution", 600, 0, 6);
91
92    TRandom3 rand; // Generatore di numeri casuali
93    rand.SetSeed(0); // Imposta il seed del generatore di numeri casuali
94
95    // Genero le particelle e riempio gli istogrammi
96    createInstances(n_event, n_particles_event, &rand, event_particles,
97                    hMassKStarDecay, hInvariantMass, hMassOppositeSign,
98                    hMassSameSign, hMassPionKaonOpposite, hMassPionKaonSame,
```

```
                  hType, hEnergy, hTheta, hPhi, hPout, hPtrasv);

  // inizializzo un array con tutti gli istogrammi
  std::array<TH1D*, 12> histograms = {hType,
                                      hInvariantMass,
                                      hMassOppositeSign,
                                      hMassSameSign,
                                      hMassPionKaonOpposite,
                                      hMassPionKaonSame,
                                      hMassKStarDecay,
                                      hEnergy,
                                      hTheta,
                                      hPhi,
                                      hPout,
                                      hPtrasv};

  //disegno gli istogrammi
  std::cout << "Drawing histograms..." << std::endl;

  canvas->cd(1);
  hType->Draw("E");
  for (int i = 1; i < 12; ++i) {
    canvas->cd(i + 1);
    histograms[i]->Draw("P");
  }

  // Salvo gli istogrammi in un file ROOT
  saveHistograms(histograms, "IstogrammiParticelle.root");

  canvas->Update(); // Aggiorna il canvas per visualizzare i grafici

  while (gROOT->GetListOfCanvases()->FindObject("c1")) {
    gSystem->ProcessEvents(); // Processo gli eventi del sistema (per esempio
                              // per chiudere la finestra)
    gSystem->Sleep(100);      // Aggiorno la finestra ogni 100 ms per evitare
                              // di bloccare il sistema
  }

  std::cout << "Canvas closed. Exiting program." << std::endl;

  return 0;
}

void randomParticlePosition(TRandom3* rand, Particle& particle) {
  double phi   = rand->Uniform(0, 2 * M_PI);
  double theta = rand->Uniform(0, M_PI);
  double pout  = rand->Exp(1);
  double px    = pout * std::sin(theta) * std::cos(phi);
  double py    = pout * std::sin(theta) * std::sin(phi);
  double pz    = pout * std::cos(theta);
```

```
149
150   particle.set_p(px, py, pz);
151 }
152
153 Particle createRandomParticle(TRandom3* rand) {
154   char name;
155   int  c = rand->Integer(200);
156   if (c < 80) {
157     name = 'Q';
158   } else if (c < 160) {
159     name = 'q';
160   } else if (c < 170) {
161     name = 'K';
162   } else if (c < 180) {
163     name = 'k';
164   } else if (c < 189) {
165     name = 'P';
166   } else if (c < 198) {
167     name = 'p';
168   } else {
169     name = '*';
170   }
171
172   Particle particle(name);
173   randomParticlePosition(rand, particle);
174   return particle;
175 }
176
177 void fillHistogram(const std::array<Particle, 120>& particle,
178                    TH1D* hInvariantMass, TH1D* hMassOppositeSign,
179                    TH1D* hMassSameSign, TH1D* hMassPionKaonOpposite,
180                    TH1D* hMassPionKaonSame, TH1D* hType, TH1D* hEnergy,
181                    TH1D* hTheta, TH1D* hPhi, TH1D* hPout, TH1D* hPtrasv,
182                    int n_particles) {
183   for (int i = 0; i < n_particles; ++i) {
184     double x     = particle[i].get_px();
185     double y     = particle[i].get_py();
186     double z     = particle[i].get_pz();
187     double r     = std::sqrt(x * x + y * y + z * z);
188     double theta = std::acos(z / r);
189     double phi   = std::atan2(y, x);
190     if (phi < 0) phi = phi + 2 * M_PI;
191
192     hType->Fill(particle[i].get_index());
193     hEnergy->Fill(particle[i].get_energy());
194
195     hTheta->Fill(theta);
196     hPhi->Fill(phi);
197     hPout->Fill(r);
198     hPtrasv->Fill(std::sqrt(x * x + y * y));
```

```
199
200     for (int j = i + 1; j < n_particles; ++j) {
201       double mass_inv = particle[i].invMass(particle[j]);
202
203       hInvariantMass->Fill(mass_inv);
204
205       // calcolo la massa invariante tra le particelle i e j di base e la metto
206       // negli istogrammi solo se le parti soddisfano le cond
207       // Istogrammi basati su carica discordante (cariche opposte)
208       if (particle[i].get_charge() * particle[j].get_charge() < 0) {
209         hMassOppositeSign->Fill(mass_inv);
210       }
211
212       // Istogrammi basati su carica concordante (cariche uguali)
213       else if (particle[i].get_charge() * particle[j].get_charge() > 0) {
214         hMassSameSign->Fill(mass_inv);
215       }
216
217       // Massa invariante tra particelle di tipo Pion+/Kaon- e Pion-/Kaon+
218       if ((particle[i].get_name() == 'Q' && particle[j].get_name() == 'k')
219           || (particle[i].get_name() == 'q' && particle[j].get_name() == 'K')
220           || (particle[i].get_name() == 'K' && particle[j].get_name() == 'q')
221           || (particle[i].get_name() == 'k' && particle[j].get_name() == 'Q')) {
222         hMassPionKaonOpposite->Fill(mass_inv);
223       }
224
225       // Massa invariante tra particelle di tipo Pion+/Kaon+ e Pion-/Kaon-
226       else if ((particle[i].get_name() == 'Q' && particle[j].get_name() == 'K')
227                || (particle[i].get_name() == 'q'
228                    && particle[j].get_name() == 'k')
229                || (particle[i].get_name() == 'K'
230                    && particle[j].get_name() == 'Q')
231                || (particle[i].get_name() == 'k'
232                    && particle[j].get_name() == 'q')) {
233         hMassPionKaonSame->Fill(mass_inv);
234       }
235     }
236   }
237 }
238
239 void createInstances(int n_event, int n_particles_event, TRandom3* rand,
240                      std::array<Particle, 120>& event_particles,
241                      TH1D* hMassKStarDecay, TH1D* hInvariantMass,
242                      TH1D* hMassOppositeSign, TH1D* hMassSameSign,
243                      TH1D* hMassPionKaonOpposite, TH1D* hMassPionKaonSame,
244                      TH1D* hType, TH1D* hEnergy, TH1D* hTheta, TH1D* hPhi,
245                      TH1D* hPout, TH1D* hPtrasv) {
246   std::cout << "Generating particles..." << std::endl;
247   for (int i = 0; i < n_event; ++i) {
248     int particle_count = 0;
```

```
249
250     for (int j = 0; j < n_particles_event; ++j) {
251       Particle new_particle = createRandomParticle(rand);
252
253       if (new_particle.get_name() == '*') {
254         char dau1_name = 0;
255         char dau2_name = 0;
256
257         switch (rand->Integer(2)) {
258         case 0:
259           dau1_name = 'Q';
260           dau2_name = 'k';
261           break;
262         case 1:
263           dau1_name = 'q';
264           dau2_name = 'K';
265           break;
266         }
267
268         event_particles[particle_count] = Particle(dau1_name);
269
270         event_particles[particle_count + 1] = Particle(dau2_name);
271
272         new_particle.decay2body(event_particles[particle_count],
273                                 event_particles[particle_count + 1]);
274
275         hMassKStarDecay->Fill(event_particles[particle_count].invMass(
276             event_particles[particle_count + 1]));
277
278         particle_count += 2;
279       } else {
280         event_particles[particle_count] = new_particle;
281
282         ++particle_count;
283       }
284     }
285
286     // riempimento istogrammi
287     fillHistogram(event_particles, hInvariantMass, hMassOppositeSign,
288                   hMassSameSign, hMassPionKaonOpposite, hMassPionKaonSame,
289                   hType, hEnergy, hTheta, hPhi, hPout, hPtrasv, particle_count);
290   }
291 }
292
293 void saveHistograms(const std::array<TH1D*, 12>& histograms,
294                     const std::string&           filename) {
295   std::cout << "Saving data in " << filename << "..." << std::endl;
296
297   // Creo un nuovo Tfile su cui scrivere gli istogrammi
298   TFile file(filename.c_str(), "RECREATE");
```

```
299   if (!file.IsOpen()) {
300     std::cerr << "Error: Could not open file " << filename << " for writing."
301               << std::endl;
302     return;
303   }
304 
305   // Loop su tutti gli istogrammi e scrivo ciascuno nel file
306   for (auto hist : histograms) {
307     if (hist) { hist->Write(); }
308   }
309 
310   // Chiudo il file
311   file.Close();
312 }
```