```cpp
// Simulation of collision events - Luca Morelli 2021

// COMPILING INSTRUCTION:
// 1) Open Root in this directory
//
// 2) (Only the first time or in order to modify ParticleType, ResonanceType or
// Particle)
//   Compile ParticleType.cpp, ResonanceType.cpp or Particle.cpp using :
//   for example       .L Particle.cpp+
//   (These files must be compiled in order: ParticleType.cpp,
//   ResonanceType.cpp, Particle.cpp)
//
// 3) Compile Simulation.cpp using:
//                     .L Simulation.cpp+
//
// 4) Run the Macro using:
//                      simulation()
//
// Results are saved in Output.root

#include "Particle.hpp"
#include "ParticleType.hpp"
#include "ResonanceType.hpp"

#include <iomanip>
#include <iostream>
#include <vector>
#include "TCanvas.h"
#include "TFile.h"
#include "TH1F.h"
#include "TRandom.h"
#include "TStyle.h"

// Instruction to auto link precompiled libraries for Root
R__LOAD_LIBRARY(ParticleType_cpp.so)
R__LOAD_LIBRARY(ResonanceType_cpp.so)
R__LOAD_LIBRARY(Particle_cpp.so)

// ProgressBar function - prints a progress bar during execution
void progressBar(double status, double max) {
  std::cout << "\033[34m" << std::fixed << std::setprecision(0)
            << status / max * 100 << "%\033[0m|";
  for (double i{0}; i != 30; ++i) {
    if (status / max < i / 30.)
      std::cout << " ";
    else
      std::cout << "\033[7;32m \033[0m";
  }
  if (status != max - 1)
    std::cout << "|\r";
  else
    std::cout << "|\n";
}

/* Simulation */

```

```
57 void simulate() {
58   // Global style settings for graph
59
60   gStyle->SetOptStat("emr");
61   gStyle->SetHistFillColor(kCyan);
62
63   // Initialization of the types of particles
64   Particle::addParticleType("Pione+", 0.13957, 1);
65   Particle::addParticleType("Pione-", 0.13957, -1);
66   Particle::addParticleType("Kaone+", 0.49367, 1);
67   Particle::addParticleType("Kaone-", 0.49367, -1);
68   Particle::addParticleType("Protone+", 0.93827, 1);
69   Particle::addParticleType("Protone-", 0.93827, -1);
70   Particle::addParticleType("K*", 0.89166, 0, 0.050);
71
72   // Initialization of vectors to contain particles
73   std::vector<Particle> genParticles;  // Particles generated from the event
74   std::vector<Particle> decParticles;  // Particles generated by decay
75   genParticles.reserve(100);
76   decParticles.reserve(20);
77
78   // Histograms initialization and style
79   TH1F* hPType{new TH1F("hPType", "Types of particles generated", 7, 0, 7)};
80   TH1F* hTheta{new TH1F("hTheta", "Theta distriubtion", 100, 0, M_PI)};
81   TH1F* hPhi{new TH1F("hPhi", "Phi distribution", 100, 0, 2 * M_PI)};
82   TH1F* hP{new TH1F("hP", "Momentum distribution", 100, 0, 5)};
83   TH1F* hPTras{
84       new TH1F("hPtras", "Trasversal Momentum distribution", 1000, 0, 5)};
85   TH1F* hEnergy{new TH1F("hTEnergy", "Energy Distribution", 1000, 0, 5)};
86   TH1F* hInvMass{new TH1F("hInvMass", "Invariant mass", 1000, 0, 5)};
87   TH1F* hInvMSame{new TH1F(
88       "hInvMSame", "Invariant mass calculated with same charge particles", 1000,
89       0, 5)};
90   TH1F* hInvMOpp{new TH1F(
91       "hInvMOpp", "Invariant mass calculated with opposite charge particles",
92       1000, 0, 5)};
93   TH1F* hInvMPKSame{
94       new TH1F("hInvMPKSame",
95                "Invariant mass calculated with same charge Kaons and Pions",
96                1000, 0, 5)};
97   TH1F* hInvMPKOpp{
98       new TH1F("hInvMPKOpp",
99                "Invariant mass calculated with opposite charge Kaons and Pions",
100               1000, 0, 5)};
101  TH1F* hInvMDec{new TH1F(
102      "hInvMDec", "Invariant mass calculated with particles from decayment",
103      1000, .5, 1.5)};
104
105  hPType->GetXaxis()->SetBinLabel(1, "Pions +");
106  hPType->GetXaxis()->SetBinLabel(2, "Pions -");
107  hPType->GetXaxis()->SetBinLabel(3, "Kaons +");
108  hPType->GetXaxis()->SetBinLabel(4, "Kaons -");
109  hPType->GetXaxis()->SetBinLabel(5, "Protons +");
110  hPType->GetXaxis()->SetBinLabel(6, "Protons -");
111  hPType->GetXaxis()->SetBinLabel(7, "K*");
112
113  hPType->SetXTitle("Particle Type");
```

```
  hPType->SetYTitle("Occurrences");
  hTheta->SetXTitle("Theta [Rad]");
  hTheta->SetYTitle("Occurrences");
  hPhi->SetXTitle("Phi [Rad]");
  hPhi->SetYTitle("Occurrences");
  hP->SetXTitle("P [GeV]");
  hP->SetYTitle("Occurrences");
  hPTras->SetXTitle("Trasversal Momentum [GeV]");
  hPTras->SetYTitle("Occurrences");
  hEnergy->SetXTitle("Energy [GeV]");
  hEnergy->SetYTitle("Occurrences");
  hInvMass->SetXTitle("Mass [GeV/C^2]");
  hInvMass->SetYTitle("Occurrences");
  hInvMOpp->SetXTitle("Mass [GeV/C^2]");
  hInvMOpp->SetYTitle("Occurrences");
  hInvMSame->SetXTitle("Mass [GeV/C^2]");
  hInvMSame->SetYTitle("Occurrences");
  hInvMPKSame->SetXTitle("Mass [GeV/C^2]");
  hInvMPKSame->SetYTitle("Occurrences");
  hInvMPKOpp->SetXTitle("Mass [GeV/C^2]");
  hInvMPKOpp->SetYTitle("Occurrences");
  hInvMDec->SetXTitle("Mass [GeV/C^2]");
  hInvMDec->SetYTitle("Occurrences");

  hInvMass->Sumw2();
  hInvMOpp->Sumw2();
  hInvMSame->Sumw2();
  hInvMPKSame->Sumw2();
  hInvMPKOpp->Sumw2();
  hInvMDec->Sumw2();

  gRandom->SetSeed();

  // Start of the 10^5 Events, 100 particles per Event
  for (int eventCount{0}; eventCount != 1E5; ++eventCount) {
    // Event
    for (int partcilesCount{0}; partcilesCount != 100; ++partcilesCount) {
      // Generation of a new Particle
      Particle newParticle{};

      // Random generation of momentum
      double phi{gRandom->Uniform(0, 2 * M_PI)};
      double theta{gRandom->Uniform(0, M_PI)};
      double modP{gRandom->Exp(1)};

      // Filling momentum Histos
      hPhi->Fill(phi);
      hTheta->Fill(theta);
      hP->Fill(modP);

      // Converting in cartesian coordinates and setting momentum
      newParticle.setP(modP * sin(theta) * cos(phi),
                       modP * sin(theta) * sin(phi), modP * cos(theta));

      // Random generation of the type of the new Particle
      double rand{gRandom->Uniform(0, 100)};

```

```
171       if (rand <= 1) {
172         newParticle.setParticle("K*");
173         // Decayment of K*
174         Particle decP1, decP2;  // Decayment results
175         // Random generation of the type of decay
176         if (gRandom->Integer(2) == 0) {
177           decP1.setParticle("Pione+");
178           decP2.setParticle("Kaone-");
179         } else {
180           decP1.setParticle("Pione-");
181           decP2.setParticle("Kaone+");
182         }
183         // Decay calculation, if decay can happen the resulting particles are
184         // added to decParticles
185         if (newParticle.decay2body(decP1, decP2) == 0) {
186           decParticles.push_back(decP1);
187           decParticles.push_back(decP2);
188         }
189       } else if (rand <= 11) {
190         if (gRandom->Integer(2) == 0)
191           newParticle.setParticle("Kaone+");
192         else
193           newParticle.setParticle("Kaone-");
194       } else if (rand <= 20) {
195         if (gRandom->Integer(2) == 0)
196           newParticle.setParticle("Protone+");
197         else
198           newParticle.setParticle("Protone-");
199       } else {
200         if (gRandom->Integer(2) == 0)
201           newParticle.setParticle("Pione+");
202         else
203           newParticle.setParticle("Pione-");
204       }
205       // Adding the new particle to genParticles
206       genParticles.push_back(newParticle);
207
208       // Adding data to histos
209       hPType->Fill(newParticle.getIndex());
210       hPTras->Fill(sqrt(newParticle.getPx() * newParticle.getPx() +
211                         newParticle.getPy() * newParticle.getPy()));
212       hEnergy->Fill(newParticle.getEnergy());
213     }
214
215     // Adding decayment results at the end of genParticles
216     genParticles.insert(genParticles.end(), decParticles.begin(),
217                         decParticles.end());
218
219     // Filling invarant mass histos with data
220     for (auto p1{genParticles.begin()}; p1 != genParticles.end(); ++p1) {
221       if (p1->getIndex() != 6) {
222         for (auto p2{p1 + 1}; p2 != genParticles.end(); ++p2) {
223           double invMass{p1->invMass(*p2)};
224           if (p2->getIndex() != 6) {
225             hInvMass->Fill(invMass);
226             if (p1->getCharge() * p2->getCharge() > 0) {
227               hInvMSame->Fill(invMass);
```

```
              if (p1->getMass() + p2->getMass() == 0.63324)
                hInvMPKSame->Fill(invMass);
            }
            if (p1->getCharge() * p2->getCharge() < 0) {
              hInvMOpp->Fill(invMass);
              if (p1->getMass() + p2->getMass() == 0.63324)
                hInvMPKOpp->Fill(invMass);
            }
          }
        }
      }
    }

    for (auto p{decParticles.begin()}; p != decParticles.end(); ++p) {
      hInvMDec->Fill(p->invMass(*(++p)));
    }

    // Clearing used vector for new Events
    genParticles.clear();
    decParticles.clear();

    progressBar(eventCount, 1E5);
  }

  // Creating a root file and writing aquired data
  TFile* output{new TFile("Output.root", "RECREATE")};
  output->cd();

  hPType->Write();
  hPhi->Write();
  hTheta->Write();
  hP->Write();
  hPTras->Write();
  hEnergy->Write();
  hInvMass->Write();
  hInvMOpp->Write();
  hInvMSame->Write();
  hInvMPKSame->Write();
  hInvMPKOpp->Write();
  hInvMDec->Write();

  output->ls();

  output->Close();
}
```