```cpp
// Header file ParticleType.hpp - Luca Morelli 2021

#ifndef PARTICLETYPE_HPP
#define PARTICLETYPE_HPP

#include <iostream>
#include <string>

class ParticleType {
  // Data Members
  std::string const name_;
  double const mass_;
  int const charge_;

 public:
  // Constructor declaration
  ParticleType(std::string name, double mass, int charge)
      : name_{name}, mass_{mass}, charge_{charge} {}

  // Member function declaration
  virtual void print() const;
  std::string const& getName() const;
  double getMass() const;
  int getCharge() const;
  virtual double getWidth() const;
};

// Operator overload
std::ostream& operator<<(std::ostream& os, ParticleType const& particleType);

#endif
```

```cpp
// Implementation of ParticleType.hpp - Luca Morelli 2021

#include "ParticleType.hpp"

#include <iostream>
#include <string>

// Member function definitions
void ParticleType::print() const {
  std::cout << '|' << name_ << "|Mass:" << mass_ << "Kg|Charge:" << charge_
            << "C|";
}

std::string const& ParticleType::getName() const { return name_; }

double ParticleType::getMass() const { return mass_; }

int ParticleType::getCharge() const { return charge_; }

double ParticleType::getWidth() const { return 0; }

// Operator overload definition
std::ostream& operator<<(std::ostream& os, ParticleType const& particleType) {
  particleType.print();
  return os;
}
```

```cpp
// Header file ResonanceType.hpp - Luca Morelli 2021

#ifndef RESONANCETYPE_HPP
#define RESONANCETYPE_HPP

#include "ParticleType.hpp"

#include <string>

// ResonanceType class inherits from ParticleType
class ResonanceType : public ParticleType {
  // Data member
  double const width_;

 public:
  //  Constructor
  ResonanceType(std::string name, double mass, int charge, double width)
      : ParticleType(name, mass, charge), width_{width} {}
  // Member functions
  void print() const;
  double getWidth() const;
};

// Operator overload
std::ostream& operator<<(std::ostream& os, ResonanceType const& resonanceType);

#endif
```

```cpp
// Implementation of ResonanceType.hpp - Luca Morelli 2021

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

#include <iostream>

// Member functions definitions
void ResonanceType::print() const {
  ParticleType::print();
  std::cout << "ResonanceWidth:" << width_ << '|';
}

double ResonanceType::getWidth() const { return width_; }

// Operator overload definition
std::ostream& operator<<(std::ostream& os, ResonanceType const& resonanceType) {
  resonanceType.print();
  return os;
}
```

```cpp
// Header file Particle.hpp - Luca Morelli 2021

#ifndef PARTICLE_HPP
#define PARTICLE_HPP

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

#include <iostream>
#include <string>
#include <vector>

class Particle {
  // Data members
  // Static
  static std::vector<ParticleType *> particleType_;  // Types of particles
  static int NParticleType_;  // Number of Types of particles
  static constexpr int maxNumParticleType{
      10};  // Max number of Types of particles
  // Non static
  int index_;
  double Px_, Py_, Pz_;

  // Private member functions
  static int findParticle(std::string pName);
  void boost(double bx, double by, double bz);

 public:
  // Constructors
  Particle() : index_{-1}, Px_{0}, Py_{0}, Pz_{0} {}  // Default
  Particle(std::string name, double Px = 0, double Py = 0, double Pz = 0);

  // Public member functions
  // Static
  static void addParticleType(std::string name, double mass, int charge,
, double width = 0);
  static void printParticleTypes();
  // Non static
  int getIndex() const;                // Returns the index of the type
  void setParticle(int index);         // Sets the type by index
  void setParticle(std::string name);  // Sets the type by name
  void printDetails() const;
  double getPx() const;
  double getPy() const;
  double getPz() const;
  double getMass() const;
  int getCharge() const;
  double getEnergy() const;
  double invMass(Particle const &particle2) const;
  void setP(double Px, double Py, double Pz);
  int decay2body(Particle &dau1, Particle &dau2) const;
};

// Operator overload declaration
std::ostream &operator<<(std::ostream &os, Particle const &particle);
#endif
```

```cpp
// Implementation of Particle.hpp - Luca Morelli 2021

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

#include <cmath>
#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>

// Initialization of static members
std::vector<ParticleType *> Particle::particleType_{};
int Particle::NParticleType_{0};

/*Member function definitions*/

// Returns the index of a particle or -1 if not found
int Particle::findParticle(std::string pName) {
  if (NParticleType_ == 0) {
    return -1;
  }
  for (int i{0}; i < NParticleType_; ++i) {
    if (particleType_[i]->getName() == pName) {
      return i;
    }
  }
  return -1;
}

// Particle constructor definition
Particle::Particle(std::string name, double Px, double Py, double Pz)
    : Px_{Px}, Py_{Py}, Pz_{Pz} {
  index_ = findParticle(name);
  if (index_ == -1) {
    std::cout << "ERROR: Particle " << name << " has still not been defined"
              << '\n';
  }
}

// Adds a new type of particle
void Particle::addParticleType(std::string name, double mass, int charge,
,                               double width) {
  if (NParticleType_ == maxNumParticleType) {
    std::cerr << "ERROR: reached maximum type number, can't add a new one\n";
  } else {
    if (findParticle(name) == -1) {
      if (width == 0) {
        particleType_.push_back(new ParticleType{name, mass, charge});
      } else {
        particleType_.push_back(new ResonanceType{name, mass, charge, width});
      }
      ++NParticleType_;
    }
  }
```

```cpp
}

// Sets the type of particle of a Particle object using the index of the type
void Particle::setParticle(int index) {
  if (index < NParticleType_ && index != -1) {
    index_ = index;
  } else {
    std::cerr << "ERROR: " << index
              << " is not a particle index alredy defined\n";
  }
}
// Sets the type of particle of a Particle object using the name of the type
void Particle::setParticle(std::string name) {
  setParticle(findParticle(name));
}

// Prints the types of particles already existing
void Particle::printParticleTypes() {
  for (auto &it : particleType_) {
    it->print();
    std::cout << '\n';
  }
}

// Prints the data of a Particle object
void Particle::printDetails() const {
  std::cout << "|Index:" << index_ << "|" << particleType_[index_]->getName()
            << "|Px:" << Px_ << "|Py:" << Py_ << "|Pz:" << Pz_ << '|';
}

// Gets data member and derived data
int Particle::getIndex() const { return index_; }
double Particle::getPx() const { return Px_; }
double Particle::getPy() const { return Py_; }
double Particle::getPz() const { return Pz_; }
double Particle::getMass() const { return particleType_[index_]->getMass(); }
int Particle::getCharge() const { return particleType_[index_]->getCharge(); }
double Particle::getEnergy() const {
  return sqrt(getMass() * getMass() + Px_ * Px_ + Py_ * Py_ + Pz_ * Pz_);
}
double Particle::invMass(Particle const &p2) const {
  double PxTot{Px_ + p2.getPx()};
  double PyTot{Py_ + p2.getPy()};
  double PzTot{Pz_ + p2.getPz()};
  return sqrt(pow(getEnergy() + p2.getEnergy(), 2) - PxTot * PxTot -
              PyTot * PyTot - PzTot * PzTot);
}
// Sets momentum vector
void Particle::setP(double Px, double Py, double Pz) {
  Px_ = Px;
  Py_ = Py;
  Pz_ = Pz;
}

// Operator Overload definition
std::ostream &operator<<(std::ostream &os, Particle const &particle) {
  particle.printDetails();
```

```
  return os;
}

// Management of the decay of a particle
int Particle::decay2body(Particle &dau1, Particle &dau2) const {
  if (getMass() == 0.0) {
    std::cout << "Decayment cannot be preformed if mass is zero\n";
    return 1;
  }

  double massMot = getMass();
  double massDau1 = dau1.getMass();
  double massDau2 = dau2.getMass();

  if (index_ > -1) {  // add width effect

    // gaussian random numbers

    float x1, x2, w, y1, y2;

    double invnum = 1. / RAND_MAX;
    do {
      x1 = 2.0 * rand() * invnum - 1.0;
      x2 = 2.0 * rand() * invnum - 1.0;
      w = x1 * x1 + x2 * x2;
    } while (w >= 1.0);

    w = sqrt((-2.0 * log(w)) / w);
    y1 = x1 * w;
    y2 = x2 * w;

    massMot += particleType_[index_]->getWidth() * y1;
  }

  if (massMot < massDau1 + massDau2) {
    std::cout << "Decayment cannot be preformed because mass is too low in "
                 "this channel\n";
    return 2;
  }

  double pout =
      sqrt(
          (massMot * massMot - (massDau1 + massDau2) * (massDau1 + massDau2)) *
          (massMot * massMot - (massDau1 - massDau2) * (massDau1 - massDau2))) /
      massMot * 0.5;

  double norm = 2 * M_PI / RAND_MAX;

  double phi = rand() * norm;
  double theta = rand() * norm * 0.5 - M_PI / 2.;
  dau1.setP(pout * sin(theta) * cos(phi), pout * sin(theta) * sin(phi),
            pout * cos(theta));
  dau2.setP(-pout * sin(theta) * cos(phi), -pout * sin(theta) * sin(phi),
            -pout * cos(theta));

  double energy = sqrt(Px_ * Px_ + Py_ * Py_ + Pz_ * Pz_ + massMot * massMot);

```

```
  double bx = Px_ / energy;
  double by = Py_ / energy;
  double bz = Pz_ / energy;

  dau1.boost(bx, by, bz);
  dau2.boost(bx, by, bz);

  return 0;
}

void Particle::boost(double bx, double by, double bz) {
  double energy = getEnergy();

  // Boost this Lorentz vector
  double b2 = bx * bx + by * by + bz * bz;
  double gamma = 1.0 / sqrt(1.0 - b2);
  double bp = bx * Px_ + by * Py_ + bz * Pz_;
  double gamma2 = b2 > 0 ? (gamma - 1.0) / b2 : 0.0;

  Px_ += gamma2 * bp * bx + gamma * bx * energy;
  Py_ += gamma2 * bp * by + gamma * by * energy;
  Pz_ += gamma2 * bp * bz + gamma * bz * energy;
}
```

```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();
}
```

```
// Analysis of data from collision events - Luca Morelli 2021

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

void analyze() {
  // Setting graphs style
  gStyle->SetOptStat("e");
  gStyle->SetOptFit(1);
  gStyle->SetFitFormat("7.6g");
  gStyle->SetHistFillColor(kCyan);
  gStyle->SetHistLineColor(kAzure + 10);

  // Opening file with generated data
  TFile* results = new TFile("Output.root", "READ");

  // Getting histos from file
  TH1F* hPType = (TH1F*)results->Get("hPType");
  TH1F* hPhi = (TH1F*)results->Get("hPhi");
  TH1F* hTheta = (TH1F*)results->Get("hTheta");
  TH1F* hP = (TH1F*)results->Get("hP");
  TH1F* hInvMass = (TH1F*)results->Get("hInvMass");
  TH1F* hInvMOpp = (TH1F*)results->Get("hInvMOpp");
  TH1F* hInvMSame = (TH1F*)results->Get("hInvMSame");
  TH1F* hInvMPKSame = (TH1F*)results->Get("hInvMPKSame");
  TH1F* hInvMPKOpp = (TH1F*)results->Get("hInvMPKOpp");
  TH1F* hInvMDec = (TH1F*)results->Get("hInvMDec");

  // Printing Particles types data and errors
  std::cout << "Pions+:" << hPType->GetBinContent(1) << "+/-"
            << hPType->GetBinError(1) << "\n"
            << "Pions-:" << hPType->GetBinContent(2) << "+/-"
            << hPType->GetBinError(2) << "\n"
            << "Kaone+:" << hPType->GetBinContent(3) << "+/-"
            << hPType->GetBinError(3) << "\n"
            << "Kaons-:" << hPType->GetBinContent(4) << "+/-"
            << hPType->GetBinError(4) << "\n"
            << "Protons+:" << hPType->GetBinContent(5) << "+/-"
            << hPType->GetBinError(5) << "\n"
            << "Protons-:" << hPType->GetBinContent(6) << "+/-"
            << hPType->GetBinError(6) << "\n"
            << "K*" << hPType->GetBinContent(7) << "+/-"
            << hPType->GetBinError(7) << "\n";

  // Canvas for distribution histos
  TCanvas* cDis = new TCanvas("cDis", "Distributions measured", 1500, 1000);
  cDis->Divide(2, 2);

  // Drawing histos and fits
  cDis->cd(1);
  hPType->DrawCopy();

  cDis->cd(3);
```

```
57  hPhi->Fit("pol0");
58  hPhi->DrawCopy();
59
60  cDis->cd(4);
61  hTheta->Fit("pol0");
62  hTheta->DrawCopy();
63
64  cDis->cd(2);
65  hP->Fit("expo");
66  hP->DrawCopy();
67
68  // Canvas for K* masses histos
69  TCanvas* cMass = new TCanvas("cMass", "K* Masses", 3000, 500);
70  cMass->Divide(3, 1);
71
72  // Analysis of invariant mass histos
73  // Two new histos are created subtracting same and oppostie charge histos
74
75  // Pions and Kaons histo
76  TH1F* hSubPK{new TH1F("hSubPK",
77                        "Subtraction of invariant mass of Kaons and Pions of "
78                        "Same and Opposite charge",
79                        1000, 0, 5)};
80
81  // Fills, fits and draws histo
82  cMass->cd(3);
83  hSubPK->Add(hInvMPKOpp, hInvMPKSame, 1, -1);
84  hSubPK->Fit("gaus", "", "", 0.5, 1.5);
85  hSubPK->SetXTitle("Mass [GeV/C^2]");
86  hSubPK->SetYTitle("Occurrences");
87  hSubPK->SetAxisRange(0.65, 1.4);
88  hSubPK->DrawCopy();
89
90  // All particles histo
91  TH1F* hSub{new TH1F(
92      "hSub",
93      "Subtraction of invariant mass of particles of Same and Opposite charge",
94      1000, 0, 5)};
95
96  // Fills, fits and draws histo
97  cMass->cd(2);
98  hSub->Add(hInvMOpp, hInvMSame, 1, -1);
99  hSub->Fit("gaus", "", "", 0.5, 1.5);
100 hSub->SetXTitle("Mass [GeV/C^2]");
101 hSub->SetYTitle("Occurrences");
102 hSub->SetAxisRange(0.65, 1.4);
103 hSub->DrawCopy();
104
105 // Fits and draws histo of invarant masses of particles created by decayment
106 cMass->cd(1);
107 hInvMDec->Fit("gaus");
108 hInvMDec->SetAxisRange(0.6, 1.2);
109 hInvMDec->SetFillColor(kCyan);
110 hInvMDec->SetLineColor(kAzure + 10);
111 hInvMDec->DrawCopy();
112
113 // Print canvases onto png files
```

```
  cMass->Print("Comparison.png");
  cDis->Print("Distributions.png");

  // Close file
  results->Close();
}
```

```cpp
// Test of ParticleType, ResonanceType and Particle classes - Luca Morelli 2021

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

#include <vector>
#include <iostream>

void newLine(){
    std::cout<<'\n';
}

int main(){
    ParticleType p1{"Pione",100.1,1};
    p1.print();
    newLine();
    ResonanceType p2{"Muone",20.3,-3,12.3};
    p2.print();
    newLine();
    std::vector<ParticleType*> pVector{new ParticleType{p1}};
    pVector.push_back(new ResonanceType{p2});
    pVector[0]->print();
    newLine();
    pVector[1]->print();
    newLine();
    Particle::printParticleTypes();
    Particle::addParticleType("Pione",100,0);
    Particle pione{"Pione",10,-20,0};
    Particle::addParticleType("Caone",110,0,10);
    Particle caone{"Caone",100,3,-.70};
    pione.setParticle("Caone");
    std::cout<<"I:"<<caone.getIndex()<<'\n';
    pione.setParticle("Pione");
    Particle::printParticleTypes();
    pione.printDetails();
    newLine();
    std::cout<<"Energy:"<<pione.getEnergy()<<'\n';
    caone.printDetails();
    newLine();
    std::cout<<"Energy:"<<caone.getEnergy()<<'\n'<<"Invariant Mass:"
<<caone.invMass(pione)<<'\n';
    pione.setP(300,200,100);
    pione.printDetails();
    newLine();
    std::cout<<pione<<'\n'<<caone<<'\n'<<p1<<'\n';
    return 0;
}
```