particle.cpp

```cpp
#include "particle.hpp"

#include "particle_type.hpp"
#include "resonance_type.hpp"

#include <cmath>   // for M_PI
#include <cstdlib> //for RAND_MAX
#include <iostream>
#include <stdexcept>

std::array<ParticleType*, Particle::max_n_particle_type_>
    Particle::particle_types_  = {nullptr, nullptr, nullptr, nullptr,
                                  nullptr, nullptr, nullptr};
int Particle::n_particle_type_ = 0;

Particle::Particle()
    : index_(-1)
    , px_(0.0)
    , py_(0.0)
    , pz_(0.0) {}

Particle::Particle(char name, double px, double py, double pz)
    : px_(px)
    , py_(py)
    , pz_(pz) {
  const int index = findParticle(name);

  if (index == -1) {
    throw std::runtime_error(
        "Particle name not found, unable to create Particle.");
  }

  index_ = index;
}
int Particle::findParticle(char name) {
  int index = -1;
  for (int i = 0; i < n_particle_type_; ++i) {
    if (particle_types_[i]->get_name() == name) {
      index = i;
      continue;
    }
  }

  return index;
}

void Particle::addParticleType(char name, double mass, int charge,
                               double width) {
```

```cpp
  if (n_particle_type_ >= max_n_particle_type_) {
    throw std::runtime_error("Array is full, cannot add more particle types.");
  }

  if (findParticle(name) != -1) {
    std::cout << "This name already belongs to another particle.\n";
    return;
  }

  ParticleType* new_particle_type;

  if (width) {
    new_particle_type = new ResonanceType(name, mass, charge, width);
  } else {
    new_particle_type = new ParticleType(name, mass, charge);
  }

  for (int i = 0; i < max_n_particle_type_; ++i) {
    if (new_particle_type == particle_types_[i]) {
      std::cout << "Cannot add the same particle with different names.";
      return;
    }
  }

  particle_types_[n_particle_type_] = new_particle_type;
  ++n_particle_type_;
}

void Particle::printParticleTypes() {
  for (int i = 0; i < n_particle_type_; ++i) { particle_types_[i]->print(); }
}

void Particle::printParticle() const {
  std::cout << '[' << index_ << "] Particle "
            << particle_types_[index_]->get_name() << "   Momentum = (" << px_
            << ", " << py_ << ", " << pz_ << ")\n";
}

int Particle::decay2body(Particle& dau1, Particle& dau2) const {
  if (get_mass() == 0.0) {
    printf("Decayment cannot be preformed if mass is zero\n");
    return 1;
  }

  double mass_mot  = get_mass();
  double mass_dau1 = dau1.get_mass();
  double mass_dau2 = dau2.get_mass();

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

```

```
    // gaussian random numbers

    float x1, x2, w, y1;

    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;

    mass_mot += particle_types_[index_]->get_width() * y1;
  }

  if (mass_mot < mass_dau1 + mass_dau2) {
    printf(
        "Decayment cannot be preformed because mass is too low in this channel\n");
    return 2;
  }

  double pout = sqrt((mass_mot * mass_mot
                      - (mass_dau1 + mass_dau2) * (mass_dau1 + mass_dau2))
                     * (mass_mot * mass_mot
                        - (mass_dau1 - mass_dau2) * (mass_dau1 - mass_dau2)))
              / mass_mot * 0.5;

  double norm = 2 * M_PI / RAND_MAX;

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

  double energy = sqrt(px_ * px_ + py_ * py_ + pz_ * pz_ + mass_mot * mass_mot);

  double bx = px_ / energy;
  double by = py_ / energy;
  double bz = pz_ / energy;

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

  return 0;
}

```

```cpp
void Particle::boost(double bx, double by, double bz) {
  double energy = get_energy();
  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;
}
```