Particle Catalogue – PHYS30762 C++ Project Report

Raul Scanlon

Department of Physics and Astronomy, University of Manchester
GitHub Tag: https://github.com/UofM-PHYS30762/project-particle-catalogue-raulscanlon/releases/tag/v.1.0.0

This project aims to develop a comprehensive particle catalogue in C++, which simulates the properties and behaviours of all known particles as described in the Standard Model. The project implements a class hierarchy for different particle types and utilises advanced C++ features such as polymorphism, smart pointers, and inheritance. The catalogue includes functionalities for particle decays, four-momentum calculations, and consistency checks, ensuring realistic physical behaviour is displayed by the particles in the catalogue.

1. INTRODUCTION

The particle catalogue project is designed to store and simulate particles from the Standard Model of particle physics. The goal is to create a class hierarchy that represents different types of particles, such as leptons, quarks, and bosons, along with their specific properties and behaviours. This project utilises advanced object-oriented programming principles in C++, including polymorphism, inheritance, and the use of smart pointers, to create a flexible and extensible system. The particle catalogue not only stores particle data but also simulates particle decays and checks for physical consistency. The Standard Model is the particle physics theory that describes three of the four known fundamental forces and classifies all known elementary particles [1]. These can be classified into fermions (spin-half particles) and bosons (integer-spin particles). Under fermions, there are leptons which include electrons, muon and tau particles, all with their respective anti-particle counterparts and (anti)neutrinos, and there are quarks, of which there are 6 flavours: up, down, charm, strange, top and bottom, all with their respective anti-particles. Under bosons, there are photons, the Higgs boson, Z bosons, W+ and W- bosons, and gluons.

2. PROJECT STRUCTURE

The project required many class definitions and many functions, each of which was called multiple times. To minimise code duplication and improve the readability of the code and to make debugging easier, interface was split, with separate header files defining individual classes and a main C++ file to run the code.

The project implements a hierarchy of classes to define each type of particle, which can be seen below in Figure 1.

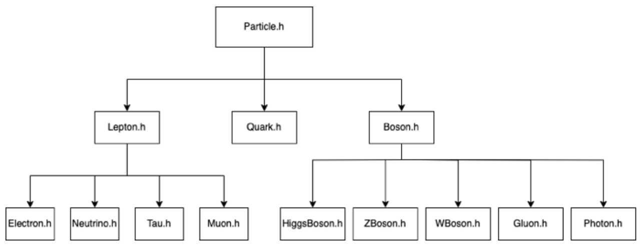


Figure 1: A UML diagram showing the class hierarchy for particles in this project.

The particle base class is abstract, defining common properties and methods for all particles. Derived classes represent particle types such as leptons, quarks and bosons, all of which define properties specific to that type of particle. Leptons and bosons are then further categorised into specific particle types, each of which also define specific particle properties such as decays or isolation or interaction variables. Each class is defined in its own header file, which is then called from the *main.cpp* file when instantiating the particles and then adds them to the particle catalogue defined in *ParticleCatalogue.h*. This file also contains printing functions to print detailed information to the screen about each particle in the catalogue.

3. PROJECT FILES

Deep copy functionality, copy and move constructors, and assignment operators are all built into each of the derived classes. Additionally, all derived classes contain overloaded print operators in order to print the information for each particle.

3.1. FourMomentum.h

This header file defines the *FourMomentum* class, which encapsulates the four-dimensional vector representing a particle's energy and momentum. The class contains methods to access each component of the four-momentum vector and defines overloaded operators to perform calculations with the four-momentum such as addition, subtraction and dot product. The class also provides a method to find the invariant mass of a particle given its four-momentum vector. Finally, the particle has an overloaded stream insertion operator to print the contents of a particle's four-momentum to the screen in a readable format.

3.2. particle.h

This header file defines the abstract base class *Particle*, which establishes a common structure and set of behaviours for all particles in the project. It has member variables of charge, spin, four-momentum, rest-mass and a Boolean to specify whether the particle is an anti-particle or not. In the case this Boolean is set to true, the charge member variable is multiplied by -1. It defines a set of pure virtual functions to be overridden by the derived particle classes, an example of this is:

```
virtual double getCharge() const = 0;.
```

The *Particle* class also defines methods to validate a particle's charge, allowing only values of ±1, 2/3, 1/3 and 0. It also contains a method to check the invariant mass of a particle, given its four-momentum (defined in the *FourMomentum* class), and outputs an error message neatly to the screen in the event of a mismatch, ensuring the physical validity of the project. The *Particle* class also defines copy and move constructors and copy and move assignment operators.

3.3. lepton.h

Lepton.h defines the *Lepton* subclass, which adds lepton-specific attributes such as lepton number and also adds a name variable to the class, making the output to the screen more readable. Like other particle classes, it also contains copy constructors, move constructors, move assignment operators and copy assignment operators, as well as a destructor for efficient memory management. The *Lepton* class also contains an overloaded print method to display lepton properties. As a base class itself, its constructors are held as protected so that they only able to be called by a derived object with access.

3.4. quark.h

This file contains the *Quark* derived class which extends quark-specific member variables to the *Particle* base class. It adds baryon number and colour charge functionality, ensuring that quarks are accurately described in the project.

3.5. boson.h

The *boson.h* file contains the *Boson* class, which is derived from the *Particle* class. This class serves to ensure the class hierarchy is physically accurate, so that the relevant boson-derived classes are inheriting from this class. This file does not add any extra functionality otherwise.

3.6. electron.h

This file defines the *Electron* class which is derived from the *Lepton* class. This adds the member variable *calorimeterEnergies* which is an array of size 4 which represents the energies deposited by an electron in a calorimeter with 4 layers. It adds setter and getter functions for the calorimeter layers as well as a check to ensure the right size input is used.

3.7. muon.h

This header file defines the *Muon* class, derived from the *Lepton* class. This class includes a Boolean member variable *isolation* and in its overloaded print operator outputs the isolation status of the muon object neatly to the screen. This functionality was required based on the script for this project.

3.8. tau.h

This file contains the *Tau* class, also derived from the *Lepton* class. This class differs from the other lepton-derived classes in that it adds functionality to handle the decay processes that tau particles undergo. The specific decay mechanism of the tau particles is underlined in the *main.cpp* file.

The decay functionality added in this file can be seen:

```
// Add a decay product

  void addDecayProduct(const
std::shared_ptr<Particle>& particle) {

    decayProducts.push_back(particle);

  }


  const std::vector<std::shared_ptr<Particle>>&
getDecayProducts() const {

    return decayProducts;

  }


  // Consistency check for decay products

  void checkDecayConsistency() const {

    double totalCharge = 0;

    for (const auto& product : decayProducts) {

      totalCharge += product->getCharge();

    }

    if (std::abs(totalCharge - getCharge()) > 1e-2) {

      throw std::runtime_error("Decay products'
charges do not sum up to the original Tau charge");

    }

  }
```

3.9. neutrino.h

This contains the *Neutrino* class, derived from the *Lepton* class. This class, similarly to the *Muon* class, contains a Boolean member variable to describe a specific property. The member variable *interaction* was not used in this project but was required in the script. The interaction status is added to the overloaded print function.

3.10. photon.h

This file describes the *Photon* class, derived from the *Boson* class. This class adds no extra functionality, other than to allow for a simple construction of a photon in the catalogue.

3.11. gluon.h

This defines the *Gluon* class, derived from the *Boson* class. It adds gluon-specific properties such as a pair of colour charges as a member variable, which are stored as 2 strings.

3.12. HiggsBoson.h

This defines the *HiggsBoson* class, derived from the *Boson* class. This file is similar to the *Tau* class in that it allows for decay functionalities to be implemented for the Higgs Boson particle in the catalogue and adds a check to ensure that the sum of decay products' charges equal the charge of the Higgs Boson. The specific decay mechanism is outlined in the '`handleDecay`' function in *main.cpp*.

3.13. WBoson.h

This defines the *WBoson* class, derived from the *Boson* class. Similarly to the *HiggsBoson* class, it allows for decay functionalities for the W Bosons and implements a check. Unlike the other boson classes, it allows the `isAntiparticle` Boolean to change the charge of the W Boson, representing the W+ and W- Bosons.

3.14. ZBoson.h

This defines the *ZBoson* class, derived from the *Boson* class. This class operates identically to the *HiggsBoson* class in its functionality but specifies the creation of a Z Boson in the catalogue.

3.15. ParticleCatalogue.h

This file defines the *ParticleCatalogue* class, which serves as an STL container for the particles created. This allows for the centralised management and manipulation of particles within the catalogue. It provides methods for adding new particles to the catalogue (via its `addParticle()` function) and printing the information of all particles. It utilises smart pointers to the base class *Particle* to allow for the polymorphic handling of particles. It contains methods for summation of the total four-momenta of all particles in the catalogue, contains a template function to get the count of a specific particle type, contains methods to get the total count of particles and methods to get the number of particles by type. It provides methods to create sub-containers of particles based on their type without the performing deep copies and contains a lambda function to sort particles by charge. Finally, the *ParticleCatalogue* class contains the `handleUserInput()` function, which asks the user for an input integer, allowing the user to decide whether to view all particles, view particles of a specific type or whether to exit the code. It also validates the user input and prompt the user to re-enter an input if an invalid input is entered. This allows for an easily digestible view of the specific particles contained in the catalogue.

The `addParticle()` function reads as following:

```
  // Add a particle to the catalogue
  void addParticle(const std::shared_ptr<Particle>&
particle) {
    particles.push_back(particle);
  }
```

The `handleUserInput()` function reads as following:

```
// User input and printing particles
  void handleUserInput() const {
    int choice;
    while (true) {
      std::cout << "Select an option:\n";
      std::cout << "1. Print all particles\n";
      std::cout << "2. Print particles by type\n";
      std::cout << "3. Exit\n";
      std::cin >> choice;
      // Check if input is valid
      if (std::cin.fail()) {
        std::cin.clear(); // Clear the error flag

std::cin.ignore(std::numeric_limits<std::streamsize>:
:max(), '\n'); // Ignore the invalid input
        std::cerr << "Invalid input. Please enter a
number.\n";
        continue; // Prompt the user again
      }
      // Handle valid choices
      switch (choice) {
      case 1:
        printAllParticles();
        break;
      case 2: {
        std::string type;
        std::cout << "Enter the particle type: ";
        std::cin >> type;
        printParticlesByType(type);
        break;
      }
      case 3:
        std::cout << "Exiting...\n";
        return; // Exit the function
      default:
        std::cerr << "Invalid choice. Please try
again.\n";
      }
    }
```

3.16. main.cpp

This file contains the main C++ code to run the program. It begins by including the necessary headers and defining the main function. This function instantiates the four-momentum of each type of particle, before instances of all particles are created with specified member variables given in the header files listed above. Then it adds these particles to the catalogue using the `catalogue.addParticle(“insert particle name”)` function specified in the *ParticleCatalogue* class.

Then, it defines the `handleDecay` function, which handles the decay of tau particles, as well as Higgs, Z, and W Bosons. It uses smart pointers to ensure efficient memory allocation. The `handleDecay` function can be seen below, but only the method for the tau decay is added for conciseness:

```
void handleDecay(const std::shared_ptr<Particle>&
particle, ParticleCatalogue& catalogue) {

  // Handle the decays of particles

  if (auto tau =
std::dynamic_pointer_cast<Tau>(particle)) {

    // Tau decays

    std::shared_ptr<Particle> decayProduct1,
decayProduct2, decayProduct3;

    std::cout << "Handling Tau decay" << std::endl;

    if (rand() % 2 == 0) {

      // Leptonic decay

      decayProduct1 = std::make_shared<Electron>(-
1.0, 0.5, 1, tau->getFourMomentum(), 0.511);

      decayProduct2 = std::make_shared<Neutrino>(0.0,
0.5, 1, tau->getFourMomentum(), 0.0, "Neutrino",
false, false);

      decayProduct3 = std::make_shared<Neutrino>(0.0,
0.5, -1, tau->getFourMomentum(), 0.0, "Anti-
Neutrino", false, true);

    } else {

      // Hadronic decay

      decayProduct1 = std::make_shared<Quark>(2.0 /
3.0, 0.5, 1.0 / 3.0, "anti-red", tau-
>getFourMomentum(), 2.3, "Anti-Up Quark", true);

      decayProduct2 = std::make_shared<Quark>(-1.0 /
3.0, 0.5, 1.0 / 3.0, "blue", tau->getFourMomentum(),
4.8, "Down Quark", false);

      decayProduct3 = std::make_shared<Neutrino>(0.0,
0.5, -1, tau->getFourMomentum(), 0.0, "Anti-
Neutrino", false, true);

    }

    std::cout << "Decay products created: " <<
std::endl;

    std::cout << "Decay product 1: " <<
decayProduct1->getTypeName() << " (Charge: " <<
decayProduct1->getCharge() << ")" << std::endl;

    std::cout << "Decay product 2: " <<
decayProduct2->getTypeName() << " (Charge: " <<
decayProduct2->getCharge() << ")" << std::endl;

    std::cout << "Decay product 3: " <<
decayProduct3->getTypeName() << " (Charge: " <<
decayProduct3->getCharge() << ")" << std::endl;

    tau->addDecayProduct(decayProduct1);

    tau->addDecayProduct(decayProduct2);

    tau->addDecayProduct(decayProduct3);

    tau->checkDecayConsistency();
```

4. HOW TO USE

The program is simple to use, with the compile line reading:

```
g++ main.cpp
```

This compile line ensures the project is run correctly, assuming the *main.cpp* file and all header files are stored in the same folder. Note that the compiler used in the development of this program is the g++-11 compiler.

5. CONSOLE OUTPUT

After compiling the *main.cpp* file and running the .exe file, the console will output that it has handled each of the tau, Higgs, W+ and W- Boson decays and will ask the user for an input:

```
Handling Tau decay

Decay products created:

Decay product 1: Quark (Charge: -0.666667)

Decay product 2: Quark (Charge: -0.333333)

Decay product 3: Anti-Neutrino (Charge: -0)

Handling Higgs decay

Decay products created:

Decay product 1: WBoson (Charge: 1)

Decay product 2: WBoson (Charge: -1)

Handling W+ Boson decay

Decay products created:

Decay product 1: Quark (Charge: 0.666667)

Decay product 2: Quark (Charge: 0.333333)

Handling W- Boson decay

Decay products created:

Decay product 1: Electron (Charge: -1)

Decay product 2: Anti-Neutrino (Charge: -0)

Handling Z Boson decay

Decay products created:

Decay product 1: Electron (Charge: -1)

Decay product 2: Electron (Charge: 1)

Select an option:

1. Print all particles

2. Print particles by type

3. Exit
```

The user would then input an integer corresponding to their desired choice. If the user enters an invalid input, the program prompts the user to re-enter a number.

In this scenario, the user inputs a letter and then upon re-entry, the user inputs a number that is not ‘1’, ‘2’, or ‘3’:

```
d

Invalid input. Please enter a number.

Select an option:

1. Print all particles

2. Print particles by type

3. Exit

6

Invalid choice. Please try again.

Select an option:

1. Print all particles

2. Print particles by type

3. Exit
```

Finally, say the user inputs ‘2’, referring to ask for a specific particle and inputs “quark” and then ‘3’ to exit the program, the following output to the console is displayed:

```
Enter the particle type: quark

Quark: Name = Up Quark, Charge = 0.666667, Spin = 0.5, Baryon Number = 0.333333, colour Charge = , Rest Mass = 2.3, Momentum = E: 2.3, px: 0, py: 0, pz: 0

Quark: Name = Anti-Up Quark, Charge = -0.666667, Spin = 0.5, Baryon Number = -0.333333, colour Charge = , Rest Mass = 2.3, Momentum = E: 2.3, px: 0, py: 0, pz: 0

Quark: Name = Down Quark, Charge = -0.333333, Spin = 0.5, Baryon Number = 0.333333, colour Charge = , Rest Mass = 4.8, Momentum = E: 4.8, px: 0, py: 0, pz: 0

Quark: Name = Anti-Down Quark, Charge = 0.333333, Spin = 0.5, Baryon Number = -0.333333, colour Charge = , Rest Mass = 4.8, Momentum = E: 4.8, px: 0, py: 0, pz: 0

Quark: Name = Charm Quark, Charge = 0.666667, Spin = 0.5, Baryon Number = 0.333333, colour Charge = , Rest Mass = 1275, Momentum = E: 1275, px: 0, py: 0, pz: 0

Quark: Name = Anti-Charm Quark, Charge = -0.666667, Spin = 0.5, Baryon Number = -0.333333, colour Charge = , Rest Mass = 1275, Momentum = E: 1275, px: 0, py: 0, pz: 0

Quark: Name = Strange Quark, Charge = -0.333333, Spin = 0.5, Baryon Number = 0.333333, colour Charge = , Rest Mass = 95, Momentum = E: 95, px: 0, py: 0, pz: 0

Quark: Name = Anti-Strange Quark, Charge = 0.333333, Spin = 0.5, Baryon Number = -0.333333, colour Charge = , Rest Mass = 95, Momentum = E: 95, px: 0, py: 0, pz: 0

Quark: Name = Top Quark, Charge = 0.666667, Spin = 0.5, Baryon Number = 0.333333, colour Charge = , Rest Mass = 173070, Momentum = E: 173070, px: 0, py: 0, pz: 0

Quark: Name = Anti-Top Quark, Charge = -0.666667, Spin = 0.5, Baryon Number = -0.333333, colour Charge = , Rest Mass = 173070, Momentum = E: 173070, px: 0, py: 0, pz: 0

Quark: Name = Bottom Quark, Charge = -0.333333, Spin = 0.5, Baryon Number = 0.333333, colour Charge = , Rest Mass = 4180, Momentum = E: 4180, px: 0, py: 0, pz: 0

Quark: Name = Anti-Bottom Quark, Charge = 0.333333, Spin = 0.5, Baryon Number = -0.333333, colour Charge = , Rest Mass = 4180, Momentum = E: 4180, px: 0, py: 0, pz: 0

Select an option:

1. Print all particles

2. Print particles by type

3. Exit

3

Exiting...

Process exited with status 0
```

This output shows the name of each quark, along with its member variables, and the output is sorted by charge.

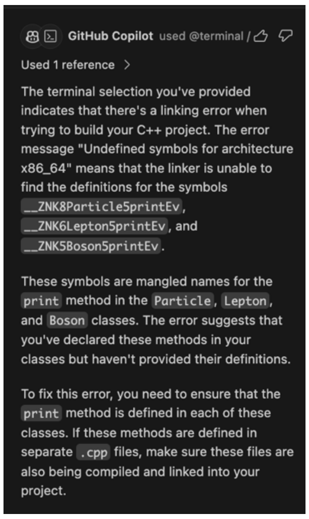
6. DISCUSSION

This project successfully creates and stores all particles described by the Standard Model in a catalogue and accurately describes the decays detailed in the project rubric. The project also successfully splits the code in its interface, allowing for better code readability, and prints its outputs to the screen in a neat fashion. This code also demonstrates the use of advanced C++ features such as smart pointers, inheritance and polymorphism. This code’s limitations are in the actual physical context of the particles it describes, and some future developments to the code could lie in the formation of some additional physical features such as matter-antimatter annihilation. Further developments to this project could be the simulation and detection of particles.

Total Word Count: 2013

ACKNOWLEDGEMENTS

GitHub Copilot was used in the development of this project, exclusively for help in debugging whilst having an error upon compilation that did not manifest itself as an error in any specific file. See below for a screenshot of the Copilot response:



The terminal selection you've provided indicates that there's a linking error when trying to build your C++ project. The error message "Undefined symbols for architecture x86_64" means that the linker is unable to find the definitions for the symbols `__ZNK8Particle5printEv`, `__ZNK6Lepton5printEv`, and `__ZNK5Boson5printEv`.

These symbols are mangled names for the `print` method in the `Particle`, `Lepton`, and `Boson` classes. The error suggests that you've declared these methods in your classes but haven't provided their definitions.

To fix this error, you need to ensure that the `print` method is defined in each of these classes. If these methods are defined in separate `.cpp` files, make sure these files are also being compiled and linked into your project.

REFERENCES

[1] R. Mann, An Introduction to Particle Physics and the Standard Model (2009).