C++ - Assignment 4

Practicing overloading special functions/ operators in classes

Charanjit Kaur & Caterina Doglioni, PHYS30762 - OOP in C++ 2023

MANCHESTER 1824
The University of Manchester

Part 1. Extend your lepton class from assignment 3

- Replace velocity with a **four-momentum P** (0.5 *mark, 0.25 if not all implemented or if problems with input checking*)
  - P = (E/c, px, py, pz) [units: MeV]
    - E cannot be negative
    - E cannot be more than the speed of light
  - Your parameterised constructor should reflect this
- Implement this four-momentum as a pointer to a dynamically allocated **std::vector** of size 4 (1 *mark, 0.5 if not dynamically allocated or no setters/getters*)
  - use new std::vector * = FourMomentum() and push_back the four-vector elements in the order above
  - because the user doesn't know what order you use, do not let user read the vector directly but instead use setters/getters for each element (getE, getpx, …)
- Add overloaded operators for sum (overload "+") that sums the four-vectors of two particles (0.5 *mark, 0.25 if attempt that doesn't do the right thing*)
- Similarly, have a function for the dot product of two particle four-vectors (call it *dotProduct*) - both this and the sum operator should take another particle as input (0.5 *mark, 0.25 if attempt that doesn't do the right thing*)

Think carefully of what to pass by reference and what to make const. We won't mark this but you can ask for feedback.

Part 2. Customize the following special functions

- (1 *mark each, 0.5 if not all implemented correctly, e.g. wrong memory management)*
  - Copy constructor
  - Copy assignment
  - Destructor
  - [challenge mark] Move constructor
  - [challenge mark] Move assignment operator
- Each of these should print out “Calling X” where X = operator/ constructor/destructor…to ease marking

Use deep copies and
check for self-assignment using this*

Part 3. Show how things work in `main()`

- (0.25 *mark for each, 0.15 if not all implemented correctly, e.g. wrong memory management)*
  - Create different particles (as before, use a “test vector” made of two electrons, four muons, one antielectron, one antimuon)
  - Sum the four-momenta of the two electrons
  - Do the dot product of the first two four-muons
  - Assignment operator of an electron to a new electron
  - Copy constructor of the first muon to a new muon
  - Move the antielectron into another antielectron using the move constructor
  - Assign the antimuon to another antimuon using the move assignment

Marks for code compilation/style

- Additional marks:
  - 0.5 for use of git (commit or tag/release)
  - 0.25 for splitting in interface and implementation
    - Add a README text file to BB / Git with the line you used to compile
    - Negative marks for not having clear code / following house style
- 1 mark will be deducted if your program produces any compilation warnings. **If your code does not compile, we will not debug/ mark it and you will get zero marks.**

Suggestion on designing/writing code

- Suggestion: don’t write all code at once, write one thing at a time
- Example:
  - start with making the particle class with only constructor and destructor, instantiate it in main()
  - compile (if it does, commit & push to git)
  - add the other data member and member function, test them in `main()`
  - compile (& commit)
- This way if something doesn’t compile by the submission deadline you still have something that compiles to submit!

Link to join the GitHub repository:

https://classroom.github.com/a/YNTicYMm