

C++ - Final project

Instructions and Q&A

Charanjit Kaur & Caterina Doglioni, PHYS30762 - OOP in C++ 2024
Credits: Niels Walet, License: CC-BY-SA-NC-04

Prelude: why/how we evaluate you

- *Beyond the grade*: we would like you to be confident when you add “C++ programming” to the skills in your CV
 - Main goal of assignments / project: practice what you have learned in the course
 - Lots of programmers hate doing input checking too, but it’s really important in real life software development
 - Mark/feedback is equivalent letting you know how you demonstrated your understanding of C++
- We also understand that it’s important for you to get good (fair) grades, so we have a prescriptive rubric and two people (a demonstrator and a course leader) marking your project
- Based on last year’s marking experience and student feedback (and the presence of ChatGPT), this year we had to be much more prescriptive for the projects so we removed the “games” and “projects suggested by students”. We may reinstate some of them for next year’s course.

Aside: job interviews

Courtesy of the internet

- *Not part of the course, so the collection is only accessible by me and visible in these lectures as an example, it's not part of the curriculum or learning objectives*
 - *How I obtained it: google / ChatGPT, so you can do the same*
- https://docs.google.com/document/d/1_SO1IIOLOuVK7_E8SoLjvSshTuBZ6dZ3AFkStKUai9g/edit

Project logistics

- Remaining assessment (50%) on project (code: 35 marks + report: 15 marks)
 - We will first have slides about the projects, and then about the report at the end
- You **cannot use external libraries** except for STL and your **code must compile on the lab computers** (same as for assignments)
 - This is because it will create problems in marking: not everyone has the same libraries installed as you
- Lab slots for using the computers will still be physically available for weeks 10,11 & 12 (with no demonstrator assistance)
 - You can ask general C++ / course questions on Teams
 - As mentioned earlier, **we will not answer specific questions** such as *“Why does my code not work”* / *“how do you want me to implement this”*
 - **No guarantee of any answers during the teaching break** (also, Caterina is on annual leave 15/03-02/04) as all this material will be covered during the next lectures and labs, we’re just pre-releasing it

Logistics - 2

- Labs are still open, but there will be no demonstrator assistance for your project
 - general code questions (not specific to implementation in the project) will be answered on Teams (preferred) or on Piazza
 - Keep in mind that in the “real world” (industry/PhD) no one will give you detailed step by step instructions, so this is the time to practice that too, and we won’t answer detailed/specific questions as we did for the assignments
 - **Don’t expect any answers between March 18th and April 2nd. Consider this a pre-release that should get you in the mindset to design this (also you have assignment 5 still going)!**
 - We will answer all questions we received in the lecture at the start of the teaching period (Week 8)
- Submission:
 - **code/github repository txt file including how to compile** should be submitted via **Blackboard**, **report** should be submitted via **Turnitin**
 - **Unlike the assignments, no Blackboard resubmissions are allowed, so only submit when ready**
- Prior to submission we will ask you to compile a quick questionnaire about the course

Plagiarism and AI

- All work should be done by you, and plagiarism is not tolerated
 - You can use your own code from previous assignments, not code from other students
 - We have everyone's code, so we will check
- According to the university's guidelines on AI
 - If you use AI for your final project you should disclose how you did it (add a paragraph to your report)
 - We reserve the right to ask you some questions in an informal oral interview after we mark your project if we need clarifications on this topic

Project details

- Choice of **3** projects - see next slides

- Literature catalogue

- Particle catalogue

- Particle detector with sub-detectors

These two projects have a detailed PDF description in addition to the slides. Read both PDF and slides, before starting to design your code or putting your hands on a keyboard!

- In addition to your code, we expect the description of functionalities, design and results in a report. They will have to be submitted separately.

- **Project deadline 12/05/2024, 23:59 (Sunday at midnight)**

- We are aware that some of you may have multiple deadlines in a short period of time, but at least you don't have to revise for this course during exam period that starts on the Monday

- Suggestion: stay on top of lectures, start thinking about project early (by Monday = start of teaching break, all material will be available), and measure your efforts (= don't overcomplicate your code)

Each project has its own git repository

- Choice of **3** projects - you'll need to join the one you choose (they are all empty)
 - Literature catalogue: <https://classroom.github.com/a/vUOavAaf>
 - Particle catalogue: <https://classroom.github.com/a/nJSYJKAp>
 - Particle detector: <https://classroom.github.com/a/LiKsGfCu>
- We expect you to have a README on how to compile your assignment, like in assignments 3-5.
- You can submit either:
 - The link to the repository (from your browser)
 - A release/tag (this is what a software developer would do)

Project idea 1: an academic literature catalogue

In this project, you don't reuse or adapt already-coded elements, but it's simpler than the other two and leaves more challenge marks to things you can think of yourself (also instructions are intentionally less detailed as compared to other projects as it is less challenging)

<https://classroom.github.com/a/vUOavAaf>

Questions about this assignment will be mostly answered by Charanjit Kaur

Literature Catalogue

- Write a program to handle a catalogue for academic literature. It should have utilities to search the literature and add literature. Such a program will use a database with the following features.
- The database should have at least 3 different types (books, thesis, journals), including at least 20 entries in total. There should be at least 2 books written by the same author(s). It should have at least the following information for each entry
 - Book: title, author(s), publisher, subject, price¹.
 - Thesis: title, author, supervisor, university name.
 - Journal: name/title, impact factor, number of volumes, editor name(s) (or number of editors), scope (e.g. PRD publishes particle physics, gravitation, field theory and cosmology papers).

¹It is not necessary to have this information in this order.

Minimal class design specifications

- Write a base class for the literature entity.
- Derived classes for each type of literature (books, theses, journals etc).
- Each of these derived types should have their specific data members. Decide carefully what goes into the base class and use virtual functions wherever appropriate. e.g.
 - Books can be written by multiple authors.
 - A thesis is associated with some university.
 - Journal has some impact factor, number of editors, and papers written by different authors (number of authors contributed to the journal).

Minimal functionality

- Use of appropriate containers to store the data.
- Load the database and show the functionality of your code
- The user interface should provide the following options:
 - Field-based search options where the field can be author, title or type (where type is book, thesis or journal). For the type-based search, the code should display all the literature of that type.
 - Inserting new entries
 - Removing some entries
 - Editing an entry

Additional features and functionality

Some suggestions:

- Use templates wherever appropriate.
- Output the data in various fancy ways.
- Use of lambda functions, try and except, namespaces, static variables and functions
- Any additional functionality other than explicitly mentioned in the project description

Rubric summary and marks assignment

- Total code marks: **35**
 - 1.Minimal class design specification and minimal functionality: **24 marks** (more details on next slides)
 - 2.Innovations: **8.5 marks** (more details on next slides)
 - Advanced code features: **4.5**
 - [challenge marks to get to 100%] Advanced functionality:[Anything additional you can think of, other than explicitly mentioned in the project description] **4.0**
 - 3.[Challenge marks to get to 100%] comments, commit history, interface and implementation: **2.5 marks**
 - Separate interface and implementation: **0.5**
 - GitHub submission with commit history: **1**
 - Good comments: **1**

Additional features and functionality marks

Challenge marks to get to 100% for advanced C++ features:

- **[2/35 marks]**: templates
- **[0.5/35 marks]**: lambda functions or static variables
- **[1/35 marks]**: exceptions (try/catch)
- **[1/35 marks]**: STL algorithms/namespaces/features for fancy output style
- 4 marks for any additional functionality in the catalogue other than explicitly mentioned in the project description. This will have to be pointed out explicitly and described in the report to be valid.

Minimal design and functionally marks

1. Base class with appropriate data members and member functions: 2
2. Complete set of derived classes: 3
3. Correct use of inheritance: 1
4. Correct use of virtual functions: 2
5. For having at least two specific data members for each type of literature: 2.5
6. For having getter/setter/print functions wherever appropriate: 2
7. Appropriate use of const (wherever applicable functions, data members etc): 1
8. Complete validation of user inputs: 1
9. Use of STL containers: 1 (0.5 for vectors and full mark for other options, in addition to challenge marks)
10. Efficient memory management: 1
11. Design of main: 0.5 (call multiple functions rather than having a single function called that calls everything else)
12. The user interface working as requested: 4.5 [correctly displaying chosen option information: 1.5, letting to add entries: 1, correctly working 'remove entry option': 1, an option to edit an entry: 1]
13. Printing the total number of entries in the selection: 0.5
14. Printing average price of a book: 0.5
15. Database as suggested: 0.5
16. Well-designed code (for the part other than points 1- 4 and 11 in this slide): 1.0

Database

- You have to prepare/get your own data file (like the course list of assignment 2) which should be part of your submission.
- You could download¹ book and theses citations from the library website and edit/update them further (or get them from any other source)
- Entries like “book price” do not necessarily need to be correct. You could just come up with some number for that.
 - Similarly, you do not need to know the total number of papers in the journal. You could just have a reasonable number for that.
- Include an identifier for each type (=book, thesis...) in the data set.
- You should have at least one enumerator (or list, or other kinds of STL containers) in the database entries, e.g. for journal scope you could have a list of disciplines.

¹See the next slide for more information related to journals.

Few example journals

Name	Impact Factor	Discipline
• Nature	65	Multidisciplinary
• Physical Review Letters (PRL)	10	Physics
• Physical Review D (PRD)	4	Particle Physics and related areas
• Journal of High Energy Physics (JHEP)	5	High Energy Physics
• Nano Research	10	Nanoscience and Nano-technology
• Nuclear Physics B	3	High Energy Physics, Particle Cosmology & Mathematical Physics
• Monthly Notices of the Royal Astronomical Society (MNRAS)	5	Astrophysics

Some of the journals are open access and some need subscription to access.

Project idea 2: a particle catalogue

In this project, you can reuse and adapt a number of elements you already coded, from assignments 3-5.

This is a set of slides that starts from the rubric, gives you hint and tells you about the project background as you keep reading. You should also read the PDF on Blackboard which has all the project background at the beginning to have the full picture of what you're doing.

Since this project is more challenging than project idea 1, the total number of marks available including challenge marks is 38 instead of 35 (you will get 100% with 35 marks and above)

<https://classroom.github.com/a/nJSYJKAp>

Questions about this assignment will be mostly answered by Caterina Doglioni

Project idea #2: a particle catalogue

- **Design and demonstrate the functionality of a catalogue based on a class hierarchy for storing data of Standard Model particles (& particles beyond it, if you'd like)**
- Core marks (approx 75% of the total project marks): we require your code to have the following (more details in next slides)
 - Particle classes in a hierarchy, using abstract classes and polymorphism, and a separate 4-momentum class
 - A particle catalogue = a customised container for particles (here you can wrap your class around existing STL containers)
 - Proper (fancy) printing functions for all particles in your catalogue
- Challenge marks to get to 100% (approx 25% of the total project marks) given by use of advanced C++ features in lectures 9 and 10, or advanced functionalities as specified

The particles you should catalogue

	mass → $\approx 2.3 \text{ MeV}/c^2$ charge → $2/3$ spin → $1/2$	mass → $\approx 1.275 \text{ GeV}/c^2$ charge → $2/3$ spin → $1/2$	mass → $\approx 173.07 \text{ GeV}/c^2$ charge → $2/3$ spin → $1/2$	mass → 0 charge → 0 spin → 1	mass → $\approx 126 \text{ GeV}/c^2$ charge → 0 spin → 0
	u up	c charm	t top	g gluon	H Higgs boson
QUARKS	mass → $\approx 4.8 \text{ MeV}/c^2$ charge → $-1/3$ spin → $1/2$	mass → $\approx 95 \text{ MeV}/c^2$ charge → $-1/3$ spin → $1/2$	mass → $\approx 4.18 \text{ GeV}/c^2$ charge → $-1/3$ spin → $1/2$	mass → 0 charge → 0 spin → 1	
	d down	s strange	b bottom	γ photon	
	mass → $0.511 \text{ MeV}/c^2$ charge → -1 spin → $1/2$	mass → $105.7 \text{ MeV}/c^2$ charge → -1 spin → $1/2$	mass → $1.777 \text{ GeV}/c^2$ charge → -1 spin → $1/2$	mass → $91.2 \text{ GeV}/c^2$ charge → 0 spin → 1	
	e electron	μ muon	τ tau	Z Z boson	
LEPTONS	mass → $< 2.2 \text{ eV}/c^2$ charge → 0 spin → $1/2$	mass → $< 0.17 \text{ MeV}/c^2$ charge → 0 spin → $1/2$	mass → $< 15.5 \text{ MeV}/c^2$ charge → 0 spin → $1/2$	mass → $80.4 \text{ GeV}/c^2$ charge → ± 1 spin → 1	GAUGE BOSONS
	ν_e electron neutrino	ν_μ muon neutrino	ν_τ tau neutrino	W W boson	

- source: wikipedia

Particle catalogue: hierarchy & design

- **Design and apply a class hierarchy for storing data of Standard Model (& beyond, if you'd like) objects**
- **Minimum hierarchy and design specification:**
 - *Abstract base class:* Abstract base class for a particle object
 - *Complete set of derived classes with “rule-of-five” for deep copying (need custom assignment & move operators, copy & move constructor, destructor):* Derived classes for specific types such as leptons, quarks, bosons...the full zoo is on [this nicely written wikipedia article](#)
 - you must have 17 separate particle classes, and have a way to instantiate these and their antiparticles (note: the Higgs boson and the photon have no antiparticles)
 - *Levels of hierarchy:* You should have at least 3 levels of hierarchy (base class <- derived classes <- derived classes)
 - *Use of virtual functions:* Decide carefully what goes into base class and use virtual functions where appropriate

Particle catalogue: hierarchy & design, rubric

- **Minimum hierarchy and design specification:**
 - *Abstract base class: [2/35 marks]* if abstract base class is present
 - *Complete set of derived classes with deep copy functionalities: [2/35 marks]* if all particles with correct antiparticles can be instantiated as different objects, more functionality can add points (see later), lack of particles reduces points proportionally to the number of missing particles
 - 💡 **hint** 💡 : you don't need to make a class for each kind of quark/lepton, how you implement it is up to you as long as all properties are present
 - *Levels of hierarchy: [2/35 marks]* for 3+ levels of hierarchy, 1 for 2, 0 for only one layer
 - Note: the hierarchy should make sense, so design before starting to code
 - *Use of virtual functions: [1/35 mark]* for proper use of virtual functions

Particle catalogue: common particle functionality

Minimum properties/functionality for all particle objects:

- *[0.5/35 marks]*: charge (follow wikipedia page) - tied to particle type, no setter needed, only getter
- *[0.5/35 marks]*: spin (follow wikipedia page) - tied to particle type, no setter needed, only getter
- four-momentum (as a separate class, evaluated separately)
 - Values should be assigned by the user of the class in main(), so it needs setters/getters
 - see later slide for functionalities of this class, going through "simpler" properties first
- *[1/35 marks]*: A function returning a string that can be used to print out all properties of the particle, including specific ones

Particle catalogue: common particle functionality

Minimum properties/functionality for some particle types:

- **[0.25/35 marks]:** lepton number, +1 for leptons and neutrinos, -1 for anti-leptons and anti-neutrinos. Tied to particle type, no setter needed, only getter.
- **[0.25/35 marks]:** quark number , +1/3 for quarks, -1/3 for antiquarks

Particle catalogue: muon, electron and neutrino functionality

Minimum functionality for particle objects, specialised:

- **[0.5/35 marks]** for **electrons**, you will need to implement a vector of energies that the electron deposited in each calorimeter layer
 - assume that a calorimeter has 4 layers, called layer_1, layer_2, layer_3, layer_4
 - **Input checking:** make sure that the total energy deposited in the calorimeter layers is equal to the energy of the electron in the 4-vector (you can use closest acceptable values if users sets something inconsistently)
- **[0.25/35 marks]** for **muons**, you will need to have the following isolation information:
 - a bool that indicates whether the muon is isolated from other particles
- **[0.25/35 marks]** for **neutrinos**, you will need to have:
 - a bool that indicates whether the neutrino has interacted or not

Particle catalogue: tau functionality

Minimum functionality for particle objects, specialised:

- *[0.8/35 marks]* for **taus**, you will need to implement:
 - A vector of pointers to the particles that it decayed into
 - *With respect to assignment 5, add the antineutrino-quark-antiquark (hadronic) decay, since now you have quark classes you can use, and remove the flag that said whether the decay was hadronic or leptonic*
 - *[0.2/35 marks]* **Consistency check**: the charges of the decaying particles should sum up to the original tau charge
- Note: for taus and other particles that decay, you don't need to add the decay particles to the particle catalogue, you should just keep the parent taus as elements of the catalogue.

Particle catalogue: quark and boson functionality

Minimum functionality for the particle objects, specialised:

- **[0.15/35 marks] for quarks** (not observable), you need to add color charge (red, green, blue or antired, antigreen, antiblue for antiparticles).
- **[0.15/35 marks] gluon**: two variables for color charges (one should be colour and one anti-colour but they can be different, same options as above)
 - **[0.2/35 marks] Consistency check**: need to make sure quarks have color, antiquarks have anticolor, gluons have both colour and anti-colour
- **[0.8/35 marks] W and Z**: vector of (smart) pointers to particles it decays into: W \rightarrow (lepton + antineutrino) or (antilepton + neutrino) or (quark + quark of different flavour); Z \rightarrow (quark + antiquark) or (lepton + antilepton) or (neutrino + antineutrino).
 - **[0.2/35 marks] Consistency check**: the charges of the decaying particles should sum up to the original boson charge
- **[0.8/35 marks] Higgs boson**: vector of (smart) pointers to particles it most commonly decays into (ZZ, WW of different sign, two photons, b quark and b antiquark)
 - **[0.2/35 marks] Consistency check**: the charges of the decaying particles should sum up to 0
- All these properties can be assigned in main() by the user of your class, so your class will need a parameterised constructor and setters/getters, and **consistency/input checks**

Particle catalogue: functionality for 4-momentum class

- **Minimum functionality for the four-momentum:**

- [1/35 marks] `std::vector` (or other container/ways to implement it) for the elements, setters and getters
 - *no need to dynamically allocate this or use a smart pointer unless you want to*
- [1/35 marks] Overloaded operators for addition, subtraction, dot product
- [1/35 marks] Function that returns a number to calculate invariant mass of four-vector (itself) using this pointer, using the formula $m = \sqrt{E^2 - \sum_i (p_i^2)}$

- **Advanced (challenge towards 100%) functionality for the four-momentum:**

- [3+3=6/35 marks] Ensure that the four-momentum components should make physical sense, go back and check your relativity lectures and/or take inspiration from <https://rivet.hepforge.org/code/2.1.1/a00224.html>
 - This kind of class will be what we test against
 - The invariant mass of the four-vector should be the rest mass of the particle that the four-vector belongs to, **will need input checking**
 - *In case it is not consistent, you should warn the user in `main()` and assign default values that make it consistent - -1 mark from the total 3 marks if this is not done*
 - *No need to add more than one system of coordinates to the physical four-vector (suggestion: stick to E, p_x, p_y, p_z)*

Particle catalogue: functionality (particle container)

- **Minimum functionality for the particle container / catalogue class**

- *[0.5/35 marks]* It should be based on (=a wrapper for) a STL container class and contain pointers of particles
 - This can contribute to the challenge marks if you use something that is not `std::vector` for its implementation
- It should have specific user-implemented methods (functions) to:
 - *[0.5/35 marks]* Get the total number of particles it contains
 - *[0.5/35 marks]* Get the number of particles of each type
 - *[0.5/35 marks]* Sum the four-momentum of all the particles inside it
 - *[0.5/35 marks]* Get a sub-container of pointers to particles of the same kind, without making deep copies (*note: think about which pointer you want to use to achieve this*)
 - *[0.5/35 marks]* Print information about one or more particles (*this should call the contained particles' print function*)
- Note: these methods should not depend on how exactly you implement the container
 - This means you can't just assume `myContainer.size()` works from `main()` - you won't get marks for the particle counting total function above

Specifications for main() program

- **Minimum functionality for the main() program**
 - **[1.5/35 marks]** Instantiate a container containing all types of particles, one per type, by either reading particle records from a file or instantiating parameterised constructor in main() *[note: don't use input from keyboard]*
 - **[1/35 marks]** Generate a full report on particle and particle characteristics contained in the container, displayed on screen
 - **[0.5/35 marks]** Show that your input checking is OK:
 - by instantiating a particle with the wrong characteristics (or non-physical four-momentum), and showing your code corrects them
 - **[1/35 marks]** Design of main(): call multiple streamlined functions from the container in main(), rather than making a container.execute() function that calls everything else

Challenge marks for all projects

- **Challenge marks to get to 100%: make use of at least three out of four of the following advanced functionalities:**
 - **[2/35 marks]:** templates
 - **[0.5/35 marks]:** lambda functions or static variables
 - **[1/35 marks]:** exceptions (try/catch)
 - **[1/35 marks]:** STL containers / algorithms (beyond `std::vector`)
- **Challenge marks to get to 100%: make sure that your commit history makes sense**
 - **[1/35 marks]:** good commit history (good messages and continuous commits as something works, rather than just a single commit/upload)
- **Challenge marks to get to 100%: split into interface and implementation**
 - **[0.5/35 marks]:** multiple files, generally one per class but short classes can go together as long as the name of the file is explanatory
- **Challenge marks to get to 100%: code comments**
 - **[1/35 marks]:** good comments throughout the code, 0.5 if attempt at comments
- This rubric is common to all projects

Negative marks

- **If your code does not compile, you will get zero marks**
- Same as assignments 3-5
- Follow our suggestion at the end of each assignment: compile and commit as you code
- Don't make things too complicated, it's better to have a simpler code that works than a complex code that does not
- **Subtract 2 marks if code is difficult to read** (e.g. variables/classes/functions are not called according to what they should be doing) or **house style not respected**
- This rubric is common to all projects

Project idea 3: a particle detector simulation

This is a more challenging project than the particle catalogue, but it's also more "marketable" to future employers if you want to show it as part of your git portfolio in the future

This is a set of slides that starts from the rubric, gives you hint and tells you about the project background as you keep reading. You should also read the PDF on Blackboard which has all the project background at the beginning to have the full picture of what you're doing.

Note: since it's more challenging than the other two project ideas, the number of total marks available including challenge marks is 40 rather than 35 (you will get 100% with 35 marks and above)

<https://classroom.github.com/a/LiKsGfCu>

Questions about this assignment will be mostly answered by Caterina Doglioni

Project idea #3: a particle detector

- **Design and demonstrate the functionality of a particle detector that is made of multiple sub-detectors**
- Core marks (approx 75% of the total project marks): we require your code to have the following (more details in next slides)
 - Detector and particle classes in two hierarchies, using abstract classes and polymorphism
 - A sub-detector container = a customised container for detector classes representing the overall detector (here you can wrap your class around existing STL containers)
 - Informative (fancy) printing functions detailing what happens to particles passing through your detectors
- Challenge marks to get to 100% (approx 25% of the total project marks) given by use of advanced C++ features in lectures 9 and 10, or advanced functionalities as specified

Particle detector: hierarchy & design

- **Design and demonstrate the functionality of a particle detector that is made of multiple sub-detectors**
- **Minimum hierarchy and design specification:**
 - **Abstract base class:** Abstract base classes for at least one of particle and sub-detector object (can be both)
 - **Complete set of derived classes with “rule-of-five” for deep copying (need custom assignment & move operators, copy & move constructor, destructor):** Derived classes for specific types of detectors mentioned later, and for a limited number of particles also mentioned later
 - **Levels of hierarchy:** You should have at least 2 levels of hierarchy for each of your particle and detector objects (base class <- derived classes)
 - **Use of virtual functions:** Decide carefully what goes into base class and use virtual functions where appropriate

Particle detector: hierarchy & design, rubric

- **Minimum hierarchy and design specification:**
 - **Abstract base class:** *[2/35 marks]* if at least one abstract base class is present
 - **Complete set of derived classes with deep copy functionalities:** *[2/35 marks]* if all 4 types of sub-detector and 4 types of particles are present, functionality adds points (see later), lack of particles reduces points proportionally to the number of missing sub-detectors/particles
 - **Levels of hierarchy:** *[2/35 marks]* for 2+ levels of hierarchy (base <- derived is 2 levels) for both particles and sub-detectors, 1 for only one of the two (particles/sub-detectors), 0 for none
 - Note: the hierarchy should make sense, so design before starting to code
 - **Use of virtual functions:** *[1/35 mark]* for proper use of virtual functions

Particle detector: common particle functionality

Minimum properties/functionality for all particle objects:

- Types of particles to be implemented: electron, muon, tau, neutrino (only one type needed for this assignment), photon, proton, neutron
- **[0.3/35 marks]:** particle identifier, an int that is useful later to keep track of what particle is passing through the detector
- **[0.3/35 marks]:** charge (follow instructions page) - tied to particle type, no setter needed, only getter
- **[0.4/35 marks]:** energy - needs both setter and getter in addition to constructor
- **[0.5/35 marks]:** A function returning a string that can be used to print out all properties of the particle, including specific ones

Particle detector: electron functionality

Minimum functionality for particle objects, electrons:

- *[1/35 marks]* for **electrons**, you will need to implement a vector of true energies that the electron could deposit in each calorimeter layer
 - assume that a calorimeter has 4 layers, called EM_1, EM_2, HAD_3, HAD_4
 - Electrons leave energy only in EM_1 and EM_2
 - **Input checking:** make sure that the total true energy deposited in the calorimeter layers is equal to the energy of the electron (you can use the closest acceptable values if users sets something inconsistently)
 - 💡 **hint** 💡 Design consideration: you can consider moving the vector of calorimeter layers to a separate class to make operations on the particles and detector easier

Particle detector: muon and neutrino functionality

Minimum functionality for particle objects, muons and neutrinos:

- *[1/35 marks]* for **muons**, you will need to have the following information:
 - a vector of bools that indicates whether the muon has interacted in the muon chamber layers INNER_LAYER and OUTER_LAYER
- *[0.5/35 marks]* for **neutrinos**, you will only need a bool with the information on whether it has interacted with the detector. It's very unlikely that this will happen, so we just set this bool to zero.

Particle detector: tau functionality

Challenge marks to get to 100%: taus

- *[2/35 challenge marks]* for **taus**, you will need to implement:
 - For leptonic decaying taus, you need a vector of smart pointers to particles that it decayed into
 - For hadronic decaying taus, you need:
 - A vector of calorimeter layers with energy deposited (the behaviour of a hadronic tau in a calorimeter is the same as that of a proton or a neutron)
 - Hadronic taus should contain a tau/antitau neutrino as this is also part of the decay
- If you don't want to attempt the challenge marks, just give it the basic properties of any particle

Particle detector: photon, proton and neutron functionality

Minimum functionality for particle objects, specialised for photons, protons and neutrons:

- *[0.5/35 marks]* similarly as for **electrons**, for each of photons, protons and neutrons you will need to implement a vector of energies that the electron deposited in each calorimeter layer
 - assume that a calorimeter has 4 layers, called EM_1, EM_2, HAD_3, HAD_4
 - Photons leave energy only in EM_1 and EM_2
 - Protons and neutrons leave energy in all four layers
 - **Input checking:** make sure that the total energy deposited in the calorimeter layers is equal to the energy of the electron (you can use the closest acceptable values if users sets something inconsistently)

Particle detector: common to all sub-detectors

Minimum properties/functionality for common sub-detector:

- *[0.5/35 marks]* Energy detected in this specific sub-detector
- *[1/35 marks]* Functionality to keep track of what particles have interacted with that sub detector (particle ID and energy)
 - The implementation can be done in different ways and we leave it up to you, but here you can use **STL containers** that aren't vectors to get challenge marks as explained in a later slide
- *Assume the detector is turned on when you use it (no on/off bool like Assignment 3)*

Particle detector: calorimeter sub-detector functionality

Minimum properties/functionality for calorimeter sub-detector:

- The calorimeter detect all particles except for muons and neutrinos, sometimes in different layers
- *[1/35 marks]* It is formed of four layers (EM_1, EM_2, HAD_3, HAD_4) where the particles release their energy
- *In the simulation of particle interaction, keep in mind that a particle may or may not leave all of its energy in one of the layers*
- *See how to set these variables when we explain the main(), as this will be different wrt Assignment 3 depending on whether you go for challenge marks or not*

Particle detector: tracker sub-detector functionality

Minimum properties/functionality for tracker sub-detector:

- The tracker only tracks charged particles, so it will not see neutrinos or neutrons or photons
- It is formed of three layers (called `inner_pixel_layer`, `pixel_layer`, `strip_layer`) where the particles can interact
- *[0.5/35 marks]*: keeping track of what layer detected a particle interaction
 - You will need to have three bool variables that denote whether a particle has interacted with one of them
 - See how to set these when we explain the `main()`
 - If ≥ 2 of these bool variables are true, then a particle has been detected, if not it escaped detection

Particle detector: muon sub-detector functionality

Minimum properties/functionality for muon sub-detector:

- The muon detector behaves like a tracker
 - 💡 **hint** 💡 *Design consideration: how do you minimise code repetition to avoid making a class that looks and behaves very close to a tracker?*
- It is formed of two layers (called `inner_muon_layer`, `outer_muon_layer`) where muons interact
 - Muons are the only particle that makes it to the muon detectors, the rest gets absorbed earlier
- **[0.5/35 marks]:** keeping track of particle interactions
 - You will need to have two `bool` variables that denote whether a muon has interacted with one of them
 - See how to set these when we explain the `main()`
 - If >1 of these variables are true, then a muon has been detected and its energy should be accounted for in the specific sub-detector variable

Particle detector: overall detector

Minimum properties/functionality for overall detector comprising all sub-detectors

- **[0.5/35 marks]** includes a container of all sub-detectors
 - *You can use an enumerator with sub-detector names to make your code clearer*
- **[0.5/35 marks]** Has a function to print out its sub-detector configuration
- **[1/35 marks]** Has a function to emulate a simplified version of the missing transverse energy: emulating a simplified version of the missing transverse energy:
 - takes as input the initial energy of a hypothetical particle-particle collision, subtracts the sum of the detected energies of all the other particles to obtain the invisible energy (*you can compare this to the neutrino's in the non-ideal simulation, see later*)
- **[1.5/35 marks]** Has a function that can determine what particle passed through its sub-detectors, see next slides for how to implement it

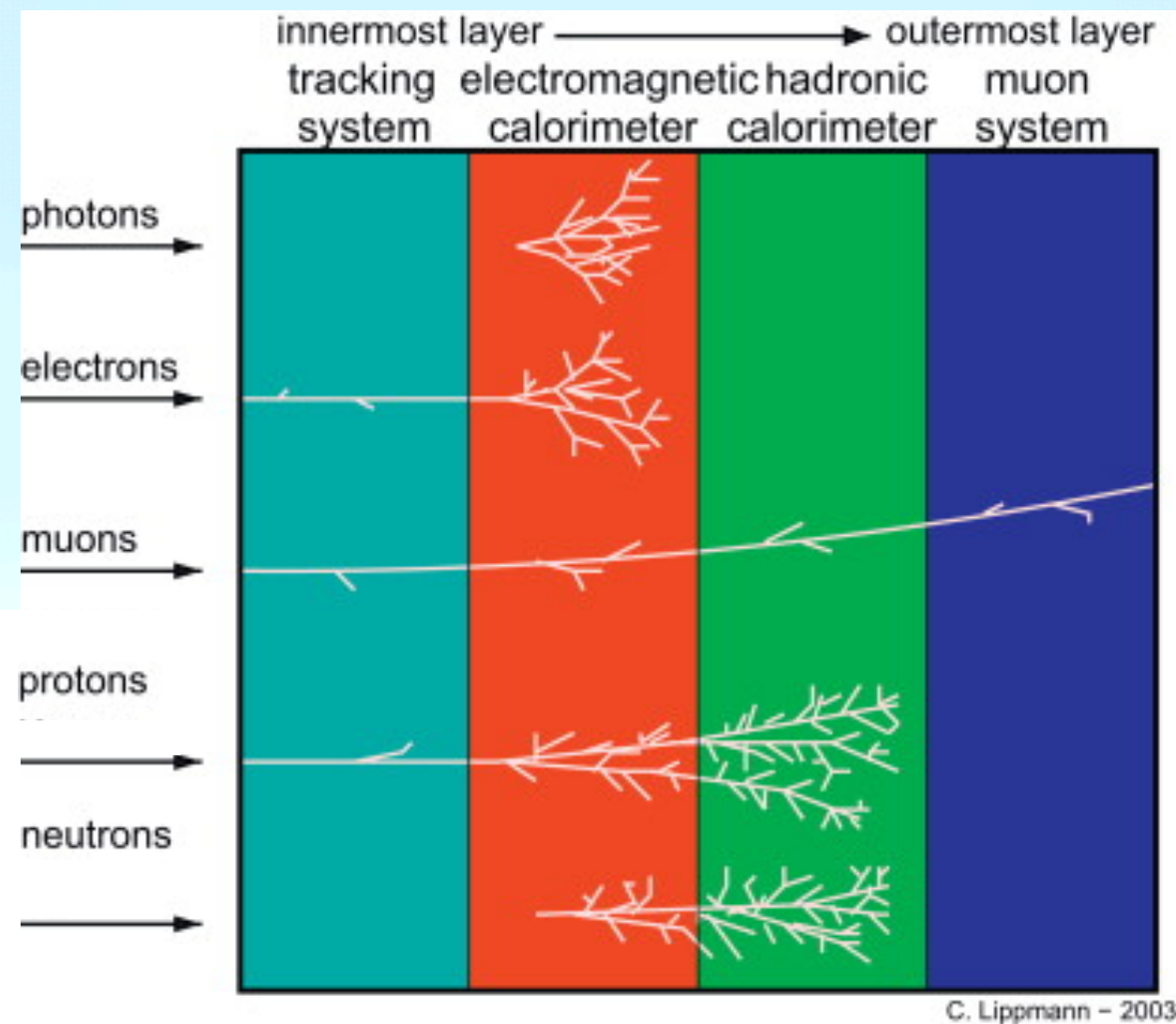
Particle detector: overall detector

How do you identify a particle? By checking where it interacts

- Photons interact in the EM calorimeter layers only
- Electrons interact in the tracker and in the EM calorimeter layers
- Muons interact in the tracker and in the muon chamber
- Taus decaying leptonically interact via their daughter particles (see above)
- Protons interact in the tracker and in all calorimeter layers
- Taus decaying hadronically interact like protons
- Neutrons interact in all calorimeter layers
- Neutrinos generally don't interact but there's an energy imbalance (invisible energy, mainly coming from the calorimeters)

Particle detector: overall detector

How do you identify a particle? By checking where it interacts



Neutrinos are not seen in the detector



If you want to know more about how a detector works overall in real life, see this talk:

https://indico.cern.ch/event/999261/contributions/4197213/attachments/2283780/3881448/DetectorParticlePhysics_1.pdf

Particle detector: main() - 1

Explanation of what we expect you to do in main()

In the main(), you will simulate a general purpose detector similar to the ones that are at the Large Hadron Collider, in the following way:

- **[0.5 mark]** Instantiate your overall detector made of a tracker, a calorimeter and a muon detector
 -  **hint**  You decide how to implement the overall detector: it can be another class, or a function...
- **[1 mark]** Create the following vectors of particles, called ‘events’ (an event is the outcome of a proton-proton interactions) together with an initial energy (energy with which the protons collided, that needs to be conserved in the initial and final state)
 - Two photons (this can happen from the decay of a Higgs boson), initial energy: 125 GeV
 - A neutron, a photon and proton (this is a common background event), initial energy: 1500 GeV
 - A tau and an anti-tau, one decaying hadronically and one decaying leptonically (this can happen from the decay of a Z boson), initial energy: 1500 GeV
 - A proton, a neutrino and an anti neutrino (this is another decay of the Z boson accompanied by a quark that is simplified as a proton, and mimics what would happen if a dark matter particle was created at the LHC), initial energy: 1500 GeV

Particle detector: main() - 2

Explanation of what we expect you to do in main()

Then pass the events through the detector

- The next slides will explain how to assign values to your sub-detector data members so you can run the simulation (challenge marks available)
- **[0.75 marks]** At the end of each event, print out the types, IDs, charge, individual energies, specific variables and sum of energies (true/detected) of all the particles that have interacted with each of the sub-detectors, and the invisible energy from the whole event (if any), and what the particle has been detected as (see example output at the end)

[0.75 mark] When you're done with all the events (you can also add more for fun if you want), you should also print:

- the total number of particles that have interacted with each of your sub-detectors
- the identification efficiency of your detector in identifying particles = number of particles correctly identified as e.g. photon/neutron/... divided by total number of particles that passed through the detector
- the energy efficiency of your detector = total detected energy / true total energy

Particle detector: simulation details

How you should select the values of the data members for the particles, standard way

Standard way (no challenge marks beyond what is in the previous slide):

- Assign the **true particle energy** to **whatever you want within the bounds of validity** (some of which were mentioned before), which are:
 - The individual particle energy should always be > 0 , e.g. using integers from 0 to the number of particles that you pass through.
 - The total visible + invisible energy of the particles in the event should be equal to the initial energy
 - The energy of the tau particles and the sum of the energy of the particles inside its decay vector should be consistent

Particle detector: simulation details

How you should select the values of the data members for the detectors, standard way

- Assign the **tracking and muon detector variables** (on whether has it interacted) as always **true**
- Assign the **energy deposited in the calorimeter layers** that is the **same as that of the electrons / photons / protons / neutrons** passing through it (remembering that electrons and photons only leave energy in EM layers and protons and neutrons leave it in all, as you need to use this information to determine which particle it is)
- If you assign the variables in this standard way, **prove in main() with some examples of incorrect inputs** at the end of the simulation that your input checking is correct
 - If it is not correct (we will also check with our own examples), you will **lose 0.2 marks per variable**

Particle detector: simulation details

How you should select the values of the data members for the particles and detector challenge marks (to get to 100%) way

[3.5/35 challenge marks] added to what is in the previous slide:

- Assign the **particle and detector characteristics** using a **random number generator**
 - 💡 **hint** 💡 The most effective way is to use the STL rand generator ([link](#)), no need to rewrite your own random number generator, but if you want you can write a wrapper class with functions that takes two objects (particle and detector) and uses the random generator to set their parameters. The advantage is readability and code streamlining, which is checked in the main() rubric.
 - The variables must still be within the bounds of validity of the previous slides, so not all of them will be independently set
 - 💡 **hint** 💡 *This is something that you have to check in the setters of the class in any case, similarly to what you did for past assignments*

Particle detector: simulation details

How you should select the values of the data members for the particles and detector, challenge marks way - practical implementation

- 💡 [3.5/35 points] hints 💡 to set the particle energies/properties:
 - The total energy of an event should be divided among the particle in the event using a random number (e.g. a number between 0 and 1 represents the fraction for the first particle and 1-that number is the fraction for the second particle)
 - How that energy would be recorded by the detectors if the detectors were fully efficient (use random number to divide through calorimeter layers)
 - Set the kind of decay that a tau will undergo, then set the particles in its decay vector (use polymorphism). You can do this by throwing a random number between 0 and 1 and asking if it's > 0.5 or not.

Particle detector: simulation details

How you should select the values of the data members for the particles and detector, challenge marks way - practical implementation

- 💡 **hints** 💡 to set the remaining detector-related variables from the detector side (here's where the simulation can become fun and more realistic...)
 - [for the tracker/muon] use random numbers to decide whether the charged particle left a track in each of the layers (this should set to be 0/false in the tracker if the particle is not charged / in the muon chamber if the particle is not a muon). Then check how many layers are true, and if it's > 2 (tracker) / > 1 (muon) then it means that the detector has detected the particle, and you can save its energy in the corresponding sub-detector variable.
 - [example for the calorimeters] use other random numbers to decide what fraction of the true energy a particle will leave in each of the calorimeter layers. You can make some of your calorimeter layers less efficient than others.
 - As a sanity check: your final energy efficiency should reflect your choice of random number generation

Particle detector: sample output, standard input (1)

- Input information:
- Event 1: two photons, total energy 125 GeV, individual energies set to 60 and 65 GeV
- Event 2: A neutron, a photon and a proton, initial energy: 1500 GeV, individual energies set to 700, 100 and 600 GeV respectively
- Way of setting variables: standard way, no random numbers. You can test whether you get the same results for cross-checking your code.
- *Output (it doesn't have to be exactly this wording, this is an example of what we are looking for - you can also make it prettier)*
- Event 1:
 - Information for particles and their interaction with the detector:
 - Photon: ID = 1, charge = 0, true total energy: 30 GeV
 - true energy in calorimeter layers: EM_1=20 GeV, EM_2=10 GeV, true energy from tracker: 0
 - Detected energy in calorimeter layers: EM_1=20 GeV, EM_2=10 GeV, true energy from tracker: 0
 - Particle detected as a photon
 - Photon: ID = 2, charge = 0, true total energy: 40 GeV,
 - true energy in calorimeter layers: EM_1=20 GeV, EM_2=10 GeV, true energy from tracker: 0
 - Detected energy in calorimeter layers: EM_1=20 GeV, EM_2=10 GeV, true energy from tracker: 0
 - Particle detected as a photon
 - True invisible energy = 0, detected invisible energy = 0.

Particle detector: sample output, standard input (2)

- Input information:
- Event 1: two photons, total energy 125 GeV, individual energies set to 60 and 65 GeV
- Event 2: A neutron, a photon and a proton, initial energy: 1500 GeV, individual energies set to 700, 100 and 600 GeV respectively
- Way of setting variables: standard way, no random numbers, so true and detected energies will be the same. You can test whether you get the same results for cross-checking your code.
- *Output, continued from previous page*
- Event 2:
 - Information for particles that have interacted with the detector:
 - Neutron: ID = 1, charge = 0, total energy = 700 GeV,
 - True energy from tracker: 0 GeV,
 - True energy in calorimeter layers: EM_1 = 50, EM_2 = 200, HAD_1 = 300, HAD_2 = 250,
 - True energy from muon detector: 0 GeV;
 - Detected energy from tracker: ...[same as true energy]
 - Particle identified as neutron
 - Photon: ID = 2, charge = 0, total energy = 100 GeV,
 - [...true and detected energies and particle types are the same as you set them, also for the next particle (proton)]
 - True invisible energy = 0, detected invisible energy = 0.

Particle detector: sample output, standard input (3)

- Input information:
- Event 1: two photons, total energy 125 GeV, individual energies set to 60 and 65 GeV
- Event 2: A neutron, a photon and a proton, initial energy: 1500 GeV, individual energies set to 700, 100 and 600 GeV respectively
- Way of setting variables: standard way, no random numbers. You can test whether you get the same results for cross-checking your code.
- *Output, continued from previous page*
- Summary after passing all particles through the detectors:
 - Total number of particles seen by the tracker: 1, energy detected by the tracker = 700 GeV;
 - Total number of particles seen by the calorimeter: 5, energy detected by the calorimeter = 1625 GeV;
 - Total number of particles seen by the muon detector: 0, energy detected by the muon detector = 0 GeV;
 - Detector identification efficiency: 100% (identified correctly/total = 5/5)
 - Energy detection: 100% (detected/true = 1625/1625)

Note: for this, you actually have to implement the correct sum/counting functions in the detector classes, not assume that the user is setting variables the standard way and “bypass it” by assigning these to the total energy of the event

Particle detector: sample output, random input

- Input information: *(using only one particle for brevity)*
- Event X: A proton: 700 GeV, tracker simulation has 0, 0 and 1 in the layers → not detected there, calorimeter simulation makes the detected energy **on average smaller** (that's how our simple random number generation works) than the true one
- Way of setting variables: random numbers. You won't be able to get the same results due to the random-ness, but use this as a sanity check of your logic - pass one particles at a time first, then expand your main() to one event, then to all events.
- Output *(it doesn't have to be exactly this wording, this is an example of what we are looking for - you can also make it prettier)*
- Event X:
 - Information for particles and their interaction with the detector:
 - Proton: ID = 1, charge = 1, energy = 700 GeV,
 - true energy from tracker: 700 GeV, true energy in calorimeter layers: EM_1 = 50 GeV, EM_2 = 300 GeV, HAD_1 = 100 GeV, HAD_2 = 250 GeV, true energy in muon detector = 0 GeV;
 - detected energy from tracker: 0 GeV, detected energy in calorimeter layers: EM_1 = 40 GeV, EM_2 = 200 GeV, HAD_1 = 50 GeV, HAD_2 = 200 GeV, true energy in muon detector = 0 GeV;
 - Particle detected as neutron (because of no energy in tracker, only in calorimeter layers)
 - Invisible energy = 0
- Summary after passing all particles through the detectors:
 - Total number of particles seen by the tracker: 0, energy detected by the tracker = 0 GeV;
 - Total number of particles seen by the calorimeter: 1, energy detected by the calorimeter = 490 GeV;
 - Total number of particles seen by the muon detector: 0, energy detected by the muon detector = 0 GeV;
 - Detector identification efficiency: 0% (particle was identified as a neutron, but it's a proton)
 - Energy detected by detectors: 70% of initial energy (490/700)

Challenge marks for all projects

- **Challenge marks to get to 100%: make use of at least three out of four of the following advanced functionalities:**
 - **[2/35 marks]:** templates
 - **[0.5/35 marks]:** lambda functions or static variables
 - **[1/35 marks]:** exceptions (try/catch)
 - **[1/35 marks]:** STL containers / algorithms (beyond `std::vector`)
- **Challenge marks to get to 100%: make sure that your commit history makes sense**
 - **[1/35 marks]:** good commit history (good messages and continuous commits as something works, rather than just a single commit/upload)
- **Challenge marks to get to 100%: split into interface and implementation**
 - **[0.5/35 marks]:** multiple files, generally one per class but short classes can go together as long as the name of the file is explanatory
- **Challenge marks to get to 100%: code comments**
 - **[1/35 marks]:** good comments throughout the code, 0.5 if attempt at comments
- This rubric is common to all projects

Negative marks

- **If your code does not compile, you will get zero marks**
- Same as assignments 3-5
- Follow our suggestion to design before you code, as said at the end of each assignment, implement one feature at a time, compile and commit as you code
- Don't make things too complicated, it's better to have a simpler code that works than a complex code that does not
- **Subtract 2 marks if code is difficult to read** (e.g. variables/classes/functions are not called according to what they should be doing) or **house style not respected**
- This rubric is common to all projects

Report

The project report (15/50 marks)

- A PDF file should be submitted on Blackboard via Turnitin (not a word document).
- What should this report cover?
 - The aim is to tell the reader **what** you did and **why** you did it the way you did (design choices)., and point them to the **how** (any additional functionalities / advanced C++ features for challenge marks should be highlighted in the report)
 - You can think about this like a lab report, where you know the aim of the experiment but you set up your own experimental procedure (your code).
- No sample report will be provided as there is no standard template to write such a report, but we give you some guidance in the next slide.

The project report (15/50 marks)

- The project report should describe the details of algorithm design, implementation and results. Specifically, it should contain the following (with an indication of the size expected)
 - Abstract (brief summary of the objectives, method and results)
 - Introduction (brief description of the project you have chosen) *~20% of length*
 - Code design and implementation. Emphasize those design elements that show the use of advanced code features; demonstrate the key ones using short code snippets *~50% of length*
 - Results (illustrates how code is used including input and output details) *~15% of length*
 - Discussion and conclusions (include discussion of how code could be improved/extended beyond project) *~10% of length*
- The total length of your report should not exceed 2500 words - you can use the word count in Overleaf or in Word. Excessive length is penalised.

Report rubric

- Three main categories:
 - Presentation (4 marks)
 - Quality of Content (6 marks)
 - Code Explanation (5 marks)
- See next slides for details

Detailed report rubric, presentation (1)

- Structure of the report (**1 mark**)
 - Could be improved in several ways (**0.3**)
 - Reasonably well structured (**0.7**)
 - Structured like a manual / documentation of a software package (**1.0**)
- Typesetting and Layout (**1 mark**)
 - Inconsistent fonts, random empty spaces, code snippets do not have formatting (**0.3**)
 - Reasonably good with only one of the issues above (**0.7**)
 - All good (**1.0**)

Detailed report rubric, presentation (2)

- Use of English (**1 mark**)
 - Several typos or major grammar problems (**0.3**)
 - Fluent, minor typos or grammar problems (**0.8**)
 - Perfect without any typos or grammar problems (**1.0**)
- Clarity and Conciseness (**1 mark**)
 - Very difficult to follow (**0.4**)
 - Reasonably clear (**0.7**)
 - Reads like a manual / documentation of a software package (**1.0**)

Detailed report rubric, quality of content (1)

- Abstract (**1 mark**)
 - Just a paragraph about the organisation of the report (**0.3**)
 - Good but lacks information about the choice of C++ features/code design (**0.6**)
 - Same quality as a public package document abstract (**1.0**)
- Introduction (**1 mark**)
 - Just a few lines (**0.4**)
 - Relevant background and outline of the project (**1.0**)

Detailed report rubric, quality of content (2)

- Flow chart or class diagram (**1.5 mark**)
 - None (**0**)
 - Reasonably clear flow chart or class diagram that could be improved (**0.7**)
 - Clear flow chart/UML with good captions (**1.0**)
 - Clear flow chart/UML with detailed captions and also referred to and explained in the text (**1.5**)
- Description of user Interface (how the user should use the classes (**1.5 mark**)
 - Missing some of the options/does not describe the code (**0.7**)
 - Covers the main point about how code should be used, but some more guidance was needed (**1.0**)
 - Professional description (**1.5**)
- References (**0.5 mark**)
- Summary and Discussion (**0.5 mark**)

Detailed report rubric, code explanation

- How code is included in the report (**1 mark**)
 - No code included (**0**)
 - Low-quality screen-shots (**0.4**)
 - Reasonably clear pieces of the code (**0.7**)
 - Professional standards (**1.0**)
- How well the included code is relevant (**1 mark**)
 - No code included (**0**)
 - Just showed some random code snippets (**0.5**)
 - Logically chosen code snippets to cover all the main points (**1.0**)
- Depth of choice of features and results of those implementations (**3 marks**)
 - Does not cover all the important (and advanced) features (**1.0**)
 - Covers all the important (and advanced) features but explanations are very simple (**2.0**)
 - Describes features well, and demonstrates in-depth knowledge of the OOP concepts (**3.0**)