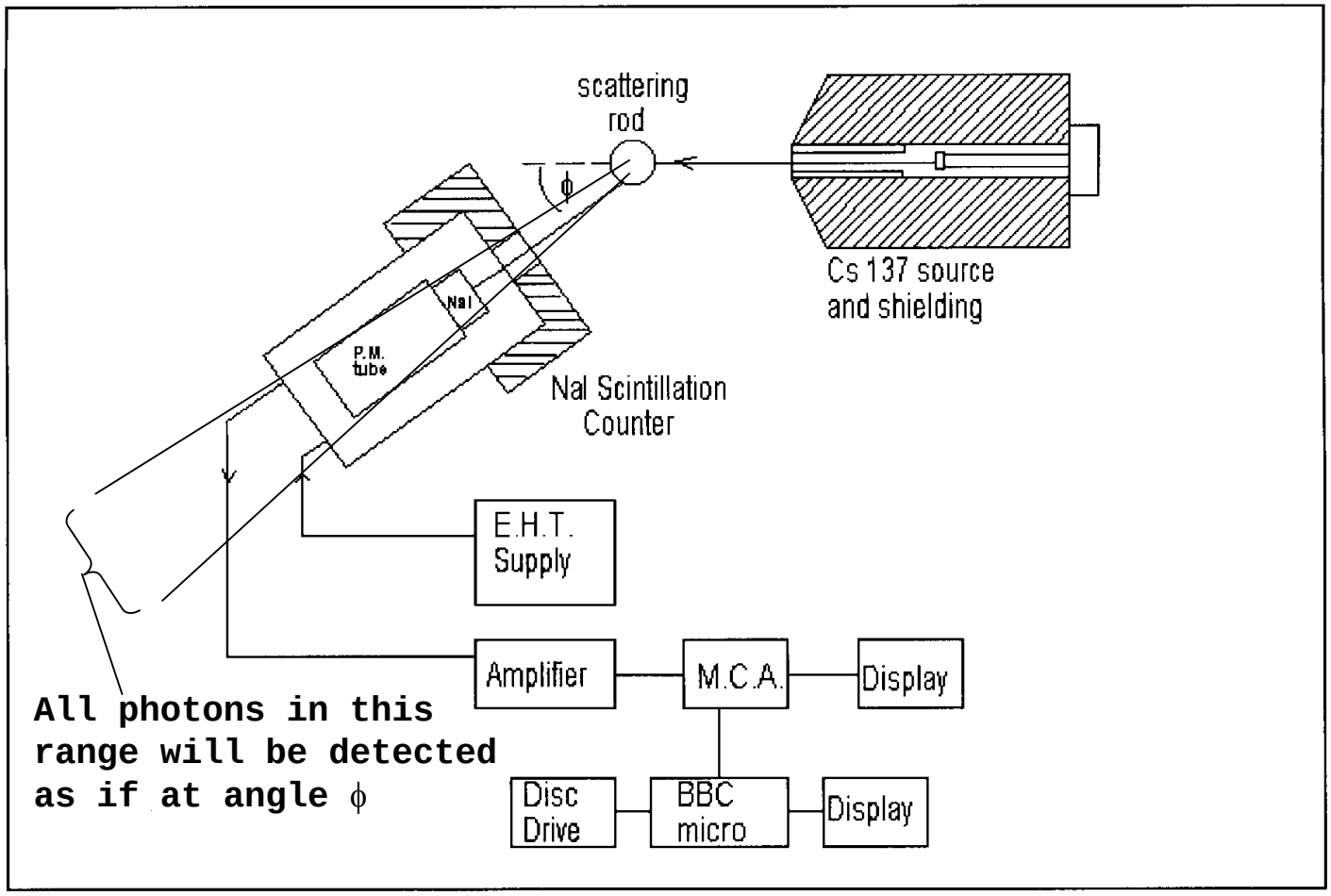
Consolidation exercise :

A "Monte Carlo" simulation of the Compton scattering experiment

Reminder of the context of the Compton scattering experiment



Problem spec:

You will write all of the classes needed to put together a "Monte Carlo" simulation of the Compton Scattering experiment.

This requires a programme which can:

- **Simulate the production and scattering of photons**
- **Simulate their detection by photmultipliers (PMT) set at a range of angles**
- **Simulate the actual width of the acceptance window of the PMT**
- **Simulate the energy resolution of the PMT**
- **Provide a realistic prediction for the rate and average energy of photons detected at each setting of the PMT.**

More detail:

In this experiment the PMT will be set at some nominal angle to detect scattered photons. However there are two effects which make the actual experimental measurements different from "theory"

1. The photomultiplier has a non zero width as depicted on the diagram.

This means that photons from quite a wide angle centred upon the nominal setting will enter the photomultiplier. These will clearly have a range of energies corresponding to their true angles of scatter. Hence the average energy measured by the photomultiplier will actually correspond to an average over this angular range.

2. The photomultiplier has a resolution of about 80 KeV.

This means that a photon entering with energy E_{true} will be detected as having an energy which has a random Gaussian distribution around E_{true} with a width of 80 KeV.

Task 1:

Try to invent the classes you think you will need to solve this problem !

Suggest you do this in groups.

You should try to come up with:

- **-class names**
- **-methods which the classes should have (including constructors)**
- **-[possibly some attributes as well]**

suggest you sketch out class declaration files (.hpp files) as you do this.

Task 2:

Write the class(es) necessary to represent a photon.

If you are following the course supplied suggestion this means you should implement the Particle class

You should take the Particle.hpp file supplied. This can be found under the ParticleGeneration button on WWW

Implement the constructor and one other methods in a file called Particle.cpp. Finish the rest in your own time

You can copy the code for most of these from the Particle.h and Particle.c files you already have from the C part of the course.

Make sure the .cpp file compiles.

If you are confident enough to have decided upon your own classes then first check with a demonstrator, and then implement whatever class you have decided on to represent a photon. This should have at least one constructor.

Produce the .hpp and .cpp files. Make sure the .cpp file compiles. **contd.⇒**
CALL A DEMONSTRATOR WHEN FINISHED TO GET THIS SIGNED OFF

Task 2 (contd)

Note:

The Particle class uses the ThreeVector class

This class has been written for you, and now has a method which returns the direction angle, phi, of the ThreeVector in the x-y plane.

You will need to download this. This can be found under the util button on the WWW pages.

Note that if you have NOT followed the directory structure which was suggested then you will need to edit the #include lines in Particle.hpp to include the ThreeVector code from wherever you put it.

Task 3

Write a class(es) which can generate a Photon Particle:

1. It should have a method which returns a Photon particle with:

-Its direction is picked randomly from the Compton scattering angular distribution

-Its energy is set to the correct Compton scattered energy appropriate to its angle.

2. This should have a constructor which takes two parameters

-the electron mass

-the incident photon energy

Write a simple test program to test your photon generator.

Comptonanalysis/
phogen.hpp
phogen.cpp

..contd..

Task 3 (contd)

If you are following the course supplied suggestion this means you should implement a class called PhoGen class.

This time you should think out and write both the .hpp and .cpp files.

Note: To help, you are supplied with a class which you can use to return to you a random value of phi picked according to the Compton scattering angular distribution. It is called `RanCom`. You can pick this up under the ComptonMC button on WWW. It is documented heavily in the .hpp and .cpp file therefore you should read it in detail to see how to use it.

If you have decided upon your own classes then first check with a demonstrator, and then implement your class(es)

Produce the .hpp and .cpp files. Make sure the .cpp file compiles

In both cases remember to write a test program to check that it works.

CALL A DEMONSTRATOR WHEN FINISHED TO GET THIS SIGNED OFF

Task 4

Write a class to represent a photomultiplier

1.This should include a constructor to allow it to be constructed to have

- **a specified angular axis**
- **an specified angular half width**
- **a specified energy resolution**

2. It should include a method to try whether a given particle would be detected within its angular range, eg:

```
void accept( Particle )
```

Note:

The Particle class is supplied in Particle.hpp and Particle.cpp under the ParticleGeneration button on WWW.

If you have invented your own class to represent particles/photons then you should use these instead.

Task 4 (contd)

3. If it is detected then the class should keep a running (internal) total of:

- **the number of Particles detected**
- **the average detected energy**

4. In the determination of the detected energy it should smear the true Particle energy with a Gaussian resolution of 80 KeV.

To do this you are supplied with a class which you can use to return to you a random value of phi picked according a Gaussian distribution with a mean of 0 and a standard deviation of 1. It is called RanGauss.

You can pick this up under the util button on WWW. It is documented heavily in the .hpp and .cpp file.

5. Finally, the class should have methods to return:

- **The total number of detected particles so far#**

- **The average energy of all the particles detected so far**

Task 5

Finally, write a main program to

- put it all together

- run it for 5000 events

- write out the final results in into a data file with cout

...