Write a program in C++ using the dev tool of your choice. Submit the appropriate program files (.cpp, .h) and documentation (.doc) file via Canvas by the date/time due. (Remember: **do not** submit zip or exe files!) Although you will want to verify the functionality of your program by creating a personal input file, the house elf will use his own file for assessment and grading.

This lab gives you a chance to develop and increase your understanding of C++ classes via the continued expansion and refinement of the previous assignments. Your program must use the following techniques and C++ features: pointers; sphere objects; a linked list ADT to hold the sphere nodes; and appropriate constructors and destructors. Also, in order to prove that you have cleaned up after yourself properly, use the Valgrind tool to check your code for memory leaks.

Style and documentation will constitute 30% of the grading for this lab. Documentation within the program is required to follow and reflect your pseudocode, which must be submitted in a separate .doc file. Therefore, please take the time necessary to make your code readable and correspondent to generally accepted coding practices, including proper division of declarations from implementation. Remember that comments are required to explain code functionality, and that these comments naturally flow from your design specification (which, in this case, is the pseudocode).

<u>Crucial:</u> Make sure your program functions correctly on the UWB Linux servers. This means that it not only compiles, but also produces correct results in Linux using the Gnu C++ tool chain. This portion constitutes 70% of the grading.

The house elf looks forward to your submission!

==

Problem statement::

Write a program that builds on the previous assignment involving the "sphere." Your implementation can begin by building on your submitted code, or you can start from scratch by completely redesigning your architecture. Refinement of the code structure is often a side effect of becoming more informed about the problem and specification evolution.

As before, the required information for the definition of the new sphere object includes center in 3-d space, radius, velocity and direction. New for this expanded version are sphere color, acceleration, mass, and the introduction of a "black hole."

The mass of a sphere will be proportional to its radius, and will induce acceleration in other spheres, and all spheres will have an acceleration component caused by the black hole.

The color component simply expands the member set of your spheres and can be recorded as an enumerated type.

All required information will be provided by the *sphere.txt* file, with all input data separated by whitespace. The information for the black hole will be given first, followed by multiple sphere specifications.

Write your program to accept initialization data for all spheres. The input data for each sphere will have the following order:

<color>

< x-coord > < y-coord > < z-coord > < radius >

< Δx > < Δy > < Δz >

where Δx, for example, is the sphere's rate of position change (=velocity) in the x direction.

Once your program begins computation (defined as t = 0 s), detect any and all collisions. A collision may be between spheres or between a sphere and the bounding container. When a sphere encounters the boundary, it is destroyed. In sphere-to-sphere collisions, defined by contact at any surface point, the smaller sphere is eliminated. Your program must run until all spheres have disappeared, accounting for each destruction event.

Note that the black hole may also eat a sphere, but only on the condition that the sphere touches the center of the black hole. We will assume a radius of 0 for the black hole in this context, so in order for it to consume a sphere, the sphere's surface must contact its given position. We will also stipulate that the black hole remain fixed in space: it will not move.

Please note that all event times must be truncated to the integer floor value for output. This will eliminate any potential rounding issues in the display of event records. (This does not, however, eliminate the requirement of double precision for all computations.)

<u>Given:</u>

- distances are measured in normalized units.

- the physical bounding box containing all spheres has the following parameters:
  Cube with origin at $(x_0,y_0,z_0) = (0,0,0)$;
  where, length x = width y = height z = 1000 units; and all enclosed points are positive (>= 0).

- minimum velocity (vector magnitude) = 10 units/s; maximum velocity (vector magnitude) = 100 units/s.

- minimum radius = 1 unit; maximum radius = 20 units.

- collisions between spheres do not alter the progress or direction of the surviving sphere.

- each sphere has mass directly proportional to its radius; for simplicity assume mass = radius.

- a single black hole exists at a specified position within the container, having 0 radius and a large effective mass also specified by the input file.

- each sphere will have a specified color; the available colors are: black, white, red, green, blue, yellow.

- black spheres, only, are immune to the effects of mass!

- in order to compute acceleration values, assume the following:
  - G =10 units/s/s, where G is the gravitational constant within the bounding container;
  - every object, including the black hole, will be treated as a point mass.

An example of the program input and output follows, where the (x,y,z) coordinates and mass of the black hole are given first:

- <u>input file *sphere.txt*</u>

```
213     821     482
2618

red
717     170     513
3
-10     5       0

black
345     416     207
18
13      -2      11

green
862     690     35
11
-16     -18     21
```

- <u>program output</u>

```
c:/>  lab2.a
```

Sphere Elimination Records
==========================

Index	Time (s)	Event type	
~~2~~	~~17~~	~~Collision~~	<- TBD
~~1~~	~~22~~	~~Collision~~	
~~3~~	~~37~~	~~Boundary~~	

*** end of run

c:/> _

<u>An additional note on the event records</u>

If there is more than one destruction event in a given time slice, then display the records in the order in which they appear in your list ADT. In other words, if sphere 7 and sphere 4 are both eliminated simultaneously, then display the event record for sphere 4 followed by that for sphere 7.