CSC3221 Practical Classes

Welcome to the practical classes for CSC3221. These classes are built to complement and drive the lectures. The practicals are very important for a programming module. Things which can seem clear during the lectures may just appear not to work at all when in front of a machine. Conversely concepts which seem intractable when described are obvious when seen.

Module Assessment

There are 3 pieces of assessment for this module:

1. Project 1 which is submitted to NESS. (20)
2. Project 2 which is submitted to NESS. (30)
3. An exam in January. (50)

Project 1 (Deadline: 4pm 7th November, 2014)

Aims

Learn to write full classes in C++ using stack and heap memory with operator overloading and consideration of class performance.

Specification

Many Computer Games are built using a Physics Engine to simulate the world of the game. Such engines simulate gravity, springs and rigid bodies as well as implementing collision detection, orientation of entities in 3 dimensions etc. A key part of most physics engines is a Vector3D (not to be confused with the vector class in the STL). Objects of this class typically represent position, velocity and acceleration of particles in the game.

Implement a class `Vector3DStack` to store x,y,z co-ordinates of an object. Store your components on the stack. Implement methods and overload operators to at least:

- return an x,y or z component of a vector
- compute the magnitude of a vector
- add, and subtract vectors
- multiply and divide a vector by a scalar
- calculate the scalar and vector product of two vectors
- overload appropriate operators to allow sensible expressions of vectors to be written (Use operator* for scalar product and operator% for vector product)
- produce a unit vector (one of magnitude 1) pointing in the same direction as a given vector
- find a unit vector orthogonal to two given vectors (hint : use vector product).

Add appropriate constructors, a copy constructor, a destructor and an assignment operator.

In a real physics engine, performance is of critical importance so try to optimise it where you can or document in comments where potential bottlenecks arise.

Rewrite the class (call it `Vector3DHeap`) so that variables are stored on the heap rather than the stack. (Note: this is probably less efficient in this scenario but serves to usefully illustrate the difference between the two representations). Test both implementations in a file `Test.cpp`

Deliverables

A zip file of the relevant project in the projects directory of Visual Studio containing the following classes:

Vector3DStack.h
Vector3DStack.cpp
Vector3DHeap.h
Vector3DHeap.cpp
Test.cpp

Mark Scheme

Required functionality Vector3DStack:	8
Required functionality Vector3DHeap:	8
Testing & Performance:	4
Total:	20