Computing and Statistical Data Analysis
Problem sheet 3
Due Monday, 31 October, 2011

**1** Look at the `TwoVector` class on the course website. The state of the vector is given by the two variables `m_x` and `m_y`, which give the $x$ and $y$ components of the vector. For this exercise you will modify and extend this class. You should write a small test program (similar to the program `TestTwoVector.cc` on the website) to show that the new class works correctly, by creating `TwoVector` objects, calling the functions, and printing the results to the monitor with `cout`. You should turn in the source code and sample output that show together the solutions for (a), (b) and (c), i.e., all three parts can be answered using the same program.

**(a)** Rewrite the `TwoVector` class so that the data members represent polar coordinates, $r$ and $\theta$. That is, get rid of `m_x` and `m_y` are replace them by variables `m_r` and `m_theta`. Rewrite the member functions so that that class behaves the same way as before (the names, return types and signatures of the member functions should be exactly the same as before).

The arguments should be interpreted the same way in both the original and new versions of the class. So, e.g., for the two-argument constructor the arguments should still be interpreted as $x$ and $y$; these are then used to set `m_r` and `m_theta`.

**(b)** In your modified `TwoVector` class, write a public member function

```
void TwoVector::reflect(TwoVector& u){  your code here }
```

such that when a `TwoVector v` calls the function,

```
v.reflect(u);
```

the effect is to reflect `v` about the line defined by the argument `u`. Show in your test program that the function works as expected.

**(c)** Overload the operators `+=` and `-=` so that they work with objects of the `TwoVector` class. Show in your test program that they work as expected. The overloaded operators `+=` and `-=` should update the state of the calling object, and also return (by reference, i.e., return type `TwoVector&`) the obdated object (i.e., `return *this;`).

**2 (optional)**

**(a)** Write a small test program containing a (long) loop in which an object or variable is created dynamically. (If you want you can create `TwoVector` objects, or better yet, some very large arrays.) Intentionally neglect to `delete` the objects at the end of the loop. Run the program and observe the behaviour. You can use the unix command `top` in another window to monitor the program's memory usage.

**(b)** Now put in the appropriate `delete` to prevent the memory leak in (a). Attempt, however, to access the pointer to the deleted object and see what happens. Finally, fix your code by setting the pointer to zero after the delete, and see what happens when you try to access the deleted object.

G. Cowan
16 October, 2011