

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.

Rewrite this class so that the data members represent polar coordinates, r and θ . That is, get rid of `m_x` and `m_y` and 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). Write a small test program to show that the new class works correctly.

(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`.)

2 Again with the `TwoVector` class (either the original or your modified version), 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`.

3 Overload the operators `+=` and `-=` so that they work with objects of the `TwoVector` class. Show that they work as expected.

4 (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.