```
1   using System;
2   using System.Collections.Generic;
3   using static Program.Constants;
4   using System.Threading;
5   using System.Threading.Tasks;
6   using System.Linq;
7   namespace Structures
8   {
9           [Serializable()]
10          public class Vector3 {
11                  // Simple 3-vector class, used for positions, velocities,
    color, etc.
12                  // setters are required for deserialization but should not be
    used outside class
13                  public double x {get; set;}
14                  public double y {get; set;}
15                  public double z {get; set;}
16                  public Vector3() {} // paramaterless constructor for
    serialization
17                  public Vector3(double x, double y, double z) {
18                          this.x = x;
19                          this.y = y;
20                          this.z = z;
21                  }
22                  // Immutable standard vectors
23                  public static readonly Vector3 zero = new Vector3(0,0,0);
24                  public static readonly Vector3 i = new Vector3(1,0,0);
25                  public static readonly Vector3 j = new Vector3(0,1,0);
26                  public static readonly Vector3 k = new Vector3(0,0,1);
27                  public override String ToString() {
28                          return $"Vector3({x},{y},{z})";
29                  }
30                  public static bool operator== (Vector3 a, Vector3 b) {
31                          // Use inherited object null equality
32                          if ((object)a == null || ((object)b == null)) return
    (object)a == null && (object)b == null;
33                          // otherwise return true if all components are within
    10^-10
34                          bool[] eq = new bool[3];
35                          for (int i = 0; i < 3; i++) {
36                                  double a1,b1;
37                                  if (i == 0) {a1 = a.x; b1 = b.x;}
38                                  else if (i == 1) {a1 = a.y; b1 = b.y;}
39                                  else {a1 = a.z; b1 = b.z;}
40                                  if (Math.Abs(a1) < 1e-2 || Math.Abs(b1) <
    1e-2) {
41                                          eq[i] = Math.Abs(a1 - b1) < 1e-10;
42                                  } else {
43                                          eq[i] = Math.Abs((a1-b1)/a1) < 1e-10
44                                               && Math.Abs((a1-b1)/b1) < 1e-10;
45                                  }
46                          }
47                          return eq[0] && eq[1] && eq[2];
48                  }
49                  public static bool operator!= (Vector3 a, Vector3 b) {
50                          // inverse of equality operator
51                          return !(a == b);
52                  }
53                  public static Vector3 operator- (Vector3 a, Vector3 b) {
54                          return new Vector3 (a.x-b.x,a.y-b.y,a.z-b.z);
55                  }
56                  public static Vector3 operator- (Vector3 a) {
57                          return new Vector3(-a.x,-a.y,-a.z);
58                  }
59                  public static Vector3 operator+ (Vector3 a, Vector3 b) {
60                          return new Vector3 (a.x+b.x,a.y+b.y,a.z+b.z);
```

```
61                  }
62                  public static Vector3 operator* (double a, Vector3 b) {
63                          return new Vector3 (a*b.x,a*b.y,a*b.z);
64                  }
65                  public static Vector3 operator/ (Vector3 a, double b) {
66                          return new Vector3 (a.x/b,a.y/b,a.z/b);
67                  }
68                  public static double dot(Vector3 a, Vector3 b) {
69                          // This could be overloaded to operator*, but an
    explicit function increases readibility.
70                          return a.x*b.x + a.y*b.y + a.z*b.z;
71                  }
72                  public static Vector3 cross(Vector3 a, Vector3 b) {
73                          return new Vector3(
74                                  a.y*b.z - a.z*b.y,
75                                  a.z*b.x - a.x*b.z,
76                                  a.x*b.y - a.y*b.x
77                          );
78                  }
79                  public static double Magnitude(Vector3 v) {
80                          // Pythagorean Theorem
81                          return Math.Sqrt(Math.Pow(v.x,2)+Math.Pow(v.y,2)
    +Math.Pow(v.z,2));
82                  }
83                  public static Vector3 Unit(Vector3 v) {
84                          // Throw exception if v is an invalid value
85                          if (v == Vector3.zero) {
86                                  throw new DivideByZeroException("Cannot take
    unit of zero vector");
87                          }
88                          return v / Vector3.Magnitude(v);
89                  }
90                  public static double UnitDot(Vector3 a, Vector3 b) {
91                          // The dot of the unit vectors
92                          return Vector3.dot(Vector3.Unit(a),Vector3.Unit(b));
93                  }
94                  public static Vector3 Log(Vector3 v, double b = Math.E) {
95                          // Polar logarithm (radius is logged, direction is
    consistent)
96                          var polar = CartesianToPolar(v);
97                          var log_polar = new Vector3 (Math.Log
    (polar.x,b),polar.y,polar.z);
98                          var log = PolarToCartesian(log_polar);
99                          return log;
100                 }
101                 public static Vector3 LogByComponent(Vector3 v, double b =
    Math.E) {
102                         // Cartesian Logarithm, all components are logged
103                         var r = new Vector3(0,0,0);
104                         if (v.x < 0) r.x = -Math.Log(-v.x,b);
105                         else if (v.x != 0) r.x = Math.Log(v.x,b);
106                         if (v.y < 0) r.y = -Math.Log(-v.y,b);
107                         else if (v.y != 0) r.y = Math.Log(v.y,b);
108                         if (v.z < 0) r.z = -Math.Log(-v.z,b);
109                         else if (v.z != 0) r.z = Math.Log(v.z,b);
110                         return r;
111                 }
112                 public static Vector3 CartesianToPolar(Vector3 v) {
113                         // ISO Convention
114                         var r = Vector3.Magnitude(v);
115                         var theta = Math.Acos(Vector3.UnitDot(v,Vector3.k));
116                         var phi = Math.Acos(Vector3.UnitDot(new Vector3
    (v.x,v.y,0),Vector3.i));
117                         if (v.y < 0) phi = -phi;
118                         return new Vector3(r,theta,phi);
119                 }
```

```
120                 public static Vector3 PolarToCartesian(Vector3 v) {
121                         // ISO Convention
122                         return Matrix3.ZRotation(v.z) * Matrix3.YRotation
    (v.y) * (v.x*Vector3.k);
123                 }
124
125         }
126         public class Matrix3 {
127                 // the fields describe the rows. Using Vector3s makes Matrix-
    Vector Multiplication
128                 // (which is the most useful operation) simpler, since then
    Vector3.dot can be used
129                 public Vector3 x {get;}
130                 public Vector3 y {get;}
131                 public Vector3 z {get;}
132                 public Matrix3(Vector3 x, Vector3 y, Vector3 z) {
133                         this.x = x;
134                         this.y = y;
135                         this.z = z;
136                 }
137                 public override String ToString() {
138                         return $"Matrix3( {x.x} {x.y} {x.z}\n         {y.x}
    {y.y} {y.z}\n         {z.x} {z.y} {z.z} )";
139                 }
140                 public static Matrix3 XRotation(double x) {
141                         return new Matrix3 (
142                                 new Vector3(1,0,0),
143                                 new Vector3(0,Math.Cos(x),Math.Sin(x)),
144                                 new Vector3(0,-Math.Sin(x),Math.Cos(x))
145                         );
146                 }
147                 public static Matrix3 YRotation(double y) {
148                         return new Matrix3 (
149                                 new Vector3(Math.Cos(y),0,Math.Sin(y)),
150                                 new Vector3(0,1,0),
151                                 new Vector3(-Math.Sin(y),0,Math.Cos(y))
152                         );
153                 }
154                 public static Matrix3 ZRotation(double z) {
155                         return new Matrix3 (
156                                 new Vector3(Math.Cos(z),-Math.Sin(z),0),
157                                 new Vector3(Math.Sin(z),Math.Cos(z),0),
158                                 new Vector3(0,0,1)
159                         );
160                 }
161                 public static Matrix3 ExtrinsicZYXRotation(double x, double
    y, double z) {
162                         return XRotation(x)*YRotation(y)*ZRotation(z);
163                 }
164                 public static Matrix3 ExtrinsicZYXRotation(Vector3 v) {
165                         return XRotation(v.x)*YRotation(v.y)*ZRotation(v.z);
166                 }
167                 public static Matrix3 IntrinsicZYXRotation(double x, double
    y, double z) {
168                         return ZRotation(z)*YRotation(y)*XRotation(x);
169                 }
170                 public static Matrix3 IntrinsicZYXRotation(Vector3 v) {
171                         return ZRotation(v.z)*YRotation(v.y)*XRotation(v.x);
172                 }
173                 public static bool operator== (Matrix3 a, Matrix3 b) {
174                         // Use vector equality
175                         return a.x == b.x && a.y == b.y && a.z == b.z;
176                 }
177                 public static bool operator!= (Matrix3 a, Matrix3 b) {
178                         return !(a == b);
179                 }
```

```
180
181                 public static Matrix3 operator+ (Matrix3 a, Matrix3 b) {
182                         // Add component-wise
183                         return new Matrix3(
184                                 a.x + b.x,
185                                 a.y + b.y,
186                                 a.z + b.z
187                         );
188                 }
189                 public static Vector3 operator* (Matrix3 m, Vector3 v) {
190                         // Using the fact that a matrix (1xn) multiplied by a
    (nx1) is equivalent to the dot of two n-vectors
191                         return new Vector3(
192                                 Vector3.dot(m.x,v),
193                                 Vector3.dot(m.y,v),
194                                 Vector3.dot(m.z,v)
195                         );
196                 }
197                 public static Matrix3 operator* (double d, Matrix3 m) {
198                         // multiply each component by d
199                         return new Matrix3(
200                                 d * m.x,
201                                 d * m.y,
202                                 d * m.z
203                         );
204                 }
205                 public static Matrix3 operator/ (Matrix3 m, double d) {
206                         // raise exception on invalid value
207                         if (d == 0) throw new DivideByZeroException("Matrix
    Division By Zero");
208                         else return (1/d) * m;
209                 }
210                 public static Matrix3 operator* (Matrix3 l, Matrix3 r) {
211                         // Finding a new matrix of the transpose of r
    converts it from row vectors to column vectors
212                         // so we can use the dot product to find each value
213                         var r_t = Matrix3.Transpose(r);
214                         return new Matrix3 (
215                                 new Vector3(
216                                         Vector3.dot(l.x,r_t.x),
217                                         Vector3.dot(l.x,r_t.y),
218                                         Vector3.dot(l.x,r_t.z)
219                                 ),
220                                 new Vector3(
221                                         Vector3.dot(l.y,r_t.x),
222                                         Vector3.dot(l.y,r_t.y),
223                                         Vector3.dot(l.y,r_t.z)
224                                 ),
225                                 new Vector3(
226                                         Vector3.dot(l.z,r_t.x),
227                                         Vector3.dot(l.z,r_t.y),
228                                         Vector3.dot(l.z,r_t.z)
229                                 )
230                         );
231                 }
232                 public static double Determinant(Matrix3 m) {
233                         return m.x.x * (m.y.y*m.z.z - m.y.z*m.z.y)
234                               -m.x.y * (m.y.x*m.z.z - m.y.z*m.z.x)
235                                  +m.x.z * (m.y.x*m.z.y - m.y.y*m.z.x);
236                 }
237                 public static Matrix3 Transpose(Matrix3 m) {
238                         return new Matrix3(
239                                 new Vector3(m.x.x,m.y.x,m.z.x),
240                                 new Vector3(m.x.y,m.y.y,m.z.y),
241                                 new Vector3(m.x.z,m.y.z,m.z.z)
242                         );
```

```
243                     }
244                     public static Matrix3 Adjugate(Matrix3 m) {
245                             return new Matrix3(
246                                     new Vector3(m.x.x,-m.y.x,m.z.x),
247                                     new Vector3(-m.x.y,m.y.y,-m.z.y),
248                                     new Vector3(m.x.z,-m.y.z,m.z.z)
249                             );
250                     }
251                     public static Matrix3 Minor(Matrix3 m) {
252                             return new Matrix3(
253                                     new Vector3(
254                                             (m.y.y*m.z.z - m.y.z*m.z.y),
255                                             (m.y.x*m.z.z - m.y.z*m.z.x),
256                                             (m.y.x*m.z.y - m.y.y*m.z.x)
257                                     ),
258                                     new Vector3(
259                                             (m.x.y*m.z.z - m.x.z*m.z.y),
260                                             (m.x.x*m.z.z - m.x.z*m.z.x),
261                                             (m.x.x*m.z.y - m.x.y*m.z.x)
262                                     ),
263                                     new Vector3(
264                                             (m.x.y*m.y.z - m.x.z*m.y.y),
265                                             (m.x.x*m.y.z - m.x.z*m.y.x),
266                                             (m.x.x*m.y.y - m.x.y*m.y.x)
267                                     )
268                             );
269                     }
270                     public static Matrix3 Inverse(Matrix3 m) {
271                             if (Matrix3.Determinant(m) == 0) throw new
    DivideByZeroException("Singular Matrix");
272                             Matrix3 A = Matrix3.Adjugate(Matrix3.Minor(m));
273                             return (1/Matrix3.Determinant(m)) * A;
274                     }
275             }
276     }
```