Lab Project 2. The Geometry Library

In this lab session, you will continue the creation of the software for the visualisation of 3D scenes. This time, you will create a library for geometric calculations in C++. This library will use the matrix library created on the last lab session to perform calculations with vectors and matrices.

You have to create at least one header file (e.g., `libgeometry.h`) containing the interface of the library. The library may have more files, but all classes and functions of the library must be created in a namespace (e.g., `libgeometry`).

The library must, at least, define the methods and classes described below. You must also create test routines for your library.

Class Quaternion: Define a quaternion whose components are of type T, where T is a class parameter.

- `Quaternion(angle, axis)`: constructs a quaternion corresponding to the rotation defined by the angle (in degrees) and the axis (class Direction) given as arguments.
- `conjugate()`: returns the conjugate of the quaternion.
- `norm()`: returns the norm of the quaternion.
- `im()`: returns the imaginary part of the quaternion.
- `inverse()`: returns the inverse of the quaternion.
- `re()`: returns the real part of the quaternion.
- `to_norm()`: returns a normalised copy of the quaternion.
- operator `+`:
  - addition of scalar and quaternion.
  - addition of quaternion and scalar.
  - addition of two quaternions.
- operator `+=`: addition (the three versions above) and assignment.
- operator `-`:
  - negation.
  - subtraction of scalar and quaternion.
  - subtraction of quaternion and scalar.
  - subtraction of two quaternions.
- operator `-=`: subtraction (the three versions above) and assignment.
- operator `*`:
  - multiplication of scalar and quaternion.
  - multiplication of quaternion and scalar.
  - multiplication of two quaternions.
- operator `*=`: multiplication (all three versions above) and assignment.
- operator `<<`: overloads operator for outputs.

Class Transform: Defines a transformation (i.e., a rotation, scaling and translation).

- `Transform()`: constructs a transform according to the arguments given. The constructor(s) must, at least, be able to create:
  - `Transform(quat)`: a rotation defined by a quaternion passed as argument.
  - `Transform(angle, axis)`: a rotation whose axis (class Direction below) and angle (in degrees) are given as arguments.
  - `Transform(vector)` a translation whose the value for each axis is given as arguments.
  - `Transform(vector)`: a scaling whose the value for each axis is given as arguments.
- `concat(transform)`: returns the concatenation of two transforms.
- `to_quat()`: returns the quaternion corresponding to a rotation stored in the transform.
- `apply(point)`:
  - returns a new point (class Point below) corresponding to the transform applied to the point given as argument.
  - returns a new direction (class Direction below) corresponding to the transform applied to the direction given as argument.
  - returns a new sphere (class Sphere below) corresponding to the transform of the sphere given as argument.
- operator `<<`: overloads operator for outputs.

Class Point: Defines a point in a N-dimensional space, where each component is of type T. T and N are class parameters. It is not necessary to redefine methods that are automatically inherited from another already defined class.

- `at(i)`: returns the i-th coordinate (i given as argument) of the point.
  Raise an exception in case of index error.
- `behind(plane)`: returns `true` if the point is behind the plane (class Plane below) given as argument, and `false` otherwise.
- `is_null()`: returns `true` if the point contains an invalid value, and `false` otherwise.
- `length_to(point)`: returns a direction (class Direction below) representing the vector between the point and another point given as argument.
- `outside(shere)`: returns `true` if the point is inside a sphere (class Sphere below) given as argument.
- `rotate(quat)`: returns a new point corresponding to the rotation of the point by the quaternion given as argument.
- operator `<<`: overloads operator for outputs.
- operator `[]`: addresses the i-th coordinate (i is given as argument) of the point. It must also permits assignment. It does not verify if i is valid.

Class Direction: Defines a direction in a N-dimensional space, where N is a class parameter. The coordinates of the direction are of type T, where T is also a class parameter. Again, it is not necessary to redefine inherited methods.

- `at(i)`: returns the i-th component (i is given as an argument) of the direction. Raises an exception if i is not a valid index.
- `is_null()`: returns `true` if the direction contains an invalid value, and `false` otherwise.
- `<<`: overloads this operator for outputs.
- `[]`: addresses the i-th component (i is given as an argument) of the direction. It must also permits assignment. It does not verify if i is valid.

Class LineSegment: Defines a line segment.

- `get_begin()`: returns the starting point (class Point) of the segment.
- `get_end()`: returns the end point (class Point) of the segment.
- `inter_coef(plane)`: returns the coefficient of intersection between the segment and a plane (class Plane below) given as an argument. The coefficient of intersection determines if the line whose the segment belongs is in front, behind or lies on the plane (class Plane below) given as argument.
- `inter(line_segment)`: returns the intersection point (class Point) between the segment and the plane (class Plane below) given as argument.
- `is_null()`: returns `true` if the segment contains an invalid value, and `false` otherwise.
- operator `<<`: overloads this operator for outputs.

Class Plane: Defines a plane.

- `is_null()`: returns `true` if the plane contains an invalid value, and `false` otherwise.
- operator `<<`: overloads this operator for outputs.

Class Sphere: Defines a sphere.

- `behind(plane)`: returns `true` if the sphere is behind a plane (class Plane) given as argument, and `false` otherwise.
- `{is_null()}`: returns `true` if the sphere contains an invalid value, and `false` otherwise.
- operator `<<`: overloads this operator for outputs.

Class Rectangle: Defines a rectangle.

- `is_null()`: returns `true` if the rectangle contains an invalid value, and `false` otherwise.
- operator `<<`: overloads this operator for outputs.

Class Triangle: Defines a triangle.

- `area()`: returns the area of the triangle.
- `is_null()`: returns `true` if the triangle contains an invalid value, and `false` otherwise.
- `get_p0()`: returns a point (class Point) corresponding to triangle's vertex 0.
- `get_p1()`: returns a point (class Point) corresponding to triangle's vertex 1.
- `get_p2()`: returns a point (class Point) corresponding to triangle's vertex 2.
- operator `<<`: overloads this operator for outputs.