University of Khartoum

Faculty of Mathematical Sciences

Department of Computer Science

Lab Manual: C++ Programming Language

Lab No (8)

8.1. In geometry, a point is a position in space. We can define a point in 3d-space as the set of coordinates x, y, and z. For example, the Point(2.0, 1.0, 0.0) would be the point at coordinate space x=2.0, y=1.0, and z=0.0.

In physics, a vector is a quantity that has a magnitude (length) and a direction (but no position). We can define a vector in 3d-space as an x, y, and z value representing the direction of the vector along the x, y, and z axis (the length can be derived from these). For example, the Vector(2.0, 0.0, 0.0) would be a vector representing a direction along the positive x-axis (only), with length 2.0.

A Vector can be applied to a Point to move the Point to a new position. This is done by adding the vector's direction to the point's position to yield a new position. For example, Point(2.0, 1.0, 0.0) + Vector(2.0, 0.0, 0.0) would yield the point (4.0, 1.0, 0.0).

Points and Vectors are often used in computer graphics (the point to represent vertices of shape, and vectors represent movement of the shape).

Given the following program:

```
#include <iostream>
class Vector3d
{
private:
double m_x, m_y, m_z;
 public:
Vector3d(double x = 0.0, double y = 0.0, double z = 0.0)
: m_x(x), m_y(y), m_z(z)
{
 }
 void print()
{std::cout << "Vector(" << m_x << " , " << m_y << " , " << m_z << ")\n";}};
 class Point3d
```

```
{
private:
double m_x, m_y, m_z;
 public:
Point3d(double x = 0.0, double y = 0.0, double z = 0.0)
: m_x(x), m_y(y), m_z(z)
{
 }
 void print()
{std::cout << "Point(" << m_x << " , " << m_y << " , " << m_z << ")\n";}
 void moveByVector(Vector3d &v){
// implement this function as a friend of class Vector3d}};
 int main()
{Point3d p(1.0, 2.0, 3.0);
Vector3d v(2.0, 2.0, -3.0);
 p.print();
p.moveByVector(v);
p.print();
 return 0;}
```

1a) Make Point3d a friend class of Vector3d, and implement function Point3d::moveByVector()

1b) Instead of making class Point3d a friend of class Vector3d, make member function Point3d::moveByVector a friend of class Vector3d.

1c) Reimplement the solution to quiz question 1b using 5 separate files: Point3d.h, Point3d.cpp, Vector3d.h, Vector3d.cpp, and main.cpp.

8.2 Execute the following code and show the output:

```
#include <iostream>
using namespace std;
 class Rectangle
{     private :
        int length;
        int width;
    public:
                void setData(int len, int wid)
        {  length = len;
            width = wid;         }
         int getArea()
        {  return length * width ;        }
                friend double getCost(Rectangle); //friend of class Rectangle };
 //friend function getCost can access private member of class
double getCost (Rectangle rect)
{     double cost;
    cost = rect.length * rect.width * 5;
    return cost; }
 int main ()
{
    Rectangle floor;
    floor.setData(20,3);
    cout << "Expense " << getCost(floor) << endl;
    return 0;
}
```

8.3 **Execute the following code and show the output:**

```
// multiple inheritance
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    Polygon (int a, int b) : width(a), height(b) {}};
class Output {
  public:
    static void print (int i);};
void Output::print (int i) {
  cout << i << '\n';}
class Rectangle: public Polygon, public Output {
  public:
    Rectangle (int a, int b) : Polygon(a,b) {}
    int area ()
      { return width*height; }};
class Triangle: public Polygon, public Output {
  public:


    Triangle (int a, int b) : Polygon(a,b) {}
    int area ()
      { return width*height/2; }};
  int main () {
  Rectangle rect (4,5);
  Triangle trgl (4,5);
  rect.print (rect.area());
  Triangle::print (trgl.area());

  return 0;}
```