**1.)**/*Write a class for instantiating the objects that represent the two-dimensional Cartesian coordinate system. A. Make a particular member function of one class as a friend function of another class for addition. B. Make the other three functions to work as a bridge between the classes for multiplication, division, and subtraction. C. Also write a small program to demonstrate that all the member functions of one class are the friend functions of another class if the former class is made friend to the latter. Make least possible classes to demonstrate all the above in a single program without conflict.*/

```
#include <iostream>
using namespace std;
class CartesianCoord;
class CoordBridge {
public:
  static CartesianCoord add(const CartesianCoord &coord1,
                            const CartesianCoord &coord2);
  static CartesianCoord subtract(const CartesianCoord &coord1,
                                 const CartesianCoord &coord2);
  static CartesianCoord multiply(const CartesianCoord &coord1,
                                 const CartesianCoord &coord2);
  static CartesianCoord divide(const CartesianCoord &coord1,
                               const CartesianCoord &coord2);
};
class CartesianCoord {
private:
  double x, y;

public:
  CartesianCoord(double x = 0, double y = 0) : x(x), y(y) {}
  friend CartesianCoord CoordBridge::add(const CartesianCoord &coord1,
                                         const CartesianCoord &coord2);
  friend CartesianCoord CoordBridge::subtract(const CartesianCoord &coord1,
                                              const CartesianCoord &coord2);
  friend CartesianCoord CoordBridge::multiply(const CartesianCoord &coord1,
                                              const CartesianCoord &coord2);
  friend CartesianCoord CoordBridge::divide(const CartesianCoord &coord1,
                                            const CartesianCoord &coord2);
  void display() const { cout << "(" << x << ", " << y << ")" << endl; }
};

CartesianCoord CoordBridge::add(const CartesianCoord &coord1,
                                const CartesianCoord &coord2) {
  CartesianCoord result;
  result.x = coord1.x + coord2.x;
  result.y = coord1.y + coord2.y;
  return result;
}

CartesianCoord CoordBridge::subtract(const CartesianCoord &coord1,
```

```
                                const CartesianCoord &coord2) {
  CartesianCoord result;
  result.x = coord1.x - coord2.x;
  result.y = coord1.y - coord2.y;
  return result;
}

CartesianCoord CoordBridge::multiply(const CartesianCoord &coord1,
                                const CartesianCoord &coord2) {
  CartesianCoord result;
  result.x = coord1.x * coord2.x;
  result.y = coord1.y * coord2.y;
  return result;
}

CartesianCoord CoordBridge::divide(const CartesianCoord &coord1,
                               const CartesianCoord &coord2) {
  CartesianCoord result;
  result.x = coord1.x / coord2.x;
  result.y = coord1.y / coord2.y;
  return result;
}

int main() {
  CartesianCoord coord1(2, 3), coord2(4, 5);

  cout << "Coord1: ";
  coord1.display();
  cout << "Coord2: ";
  coord2.display();

  cout << "Addition: ";
  CartesianCoord sum = CoordBridge::add(coord1, coord2);
  sum.display();

  cout << "Subtraction: ";
  CartesianCoord diff = CoordBridge::subtract(coord1, coord2);
  diff.display();

  cout << "Multiplication: ";
  CartesianCoord prod = CoordBridge::multiply(coord1, coord2);
  prod.display();

  cout << "Division: ";
  CartesianCoord quot = CoordBridge::divide(coord1, coord2);
  quot.display();

  return 0;
```

```
}
```
**Output**
```
Coord1: (2, 3)
Coord2: (4, 5)
Addition: (6, 8)
Subtraction: (-2, -2)
Multiplication: (8, 15)
Division: (0.5, 0.6)

2.)
/*Write a class to store x, y, and z coordinates of a point in three-dimensional
 * space. Overload addition and subtraction operators for addition and
 * subtraction of two coordinate objects. Implement the operator functions as
 * non-member functions (friend operator functions).*/
#include <iostream>
using namespace std;
class Point3D {
private:
  double x, y, z;

public:
  Point3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

  // Overloaded operators as friend functions
  friend Point3D operator+(const Point3D &lhs, const Point3D &rhs);
  friend Point3D operator-(const Point3D &lhs, const Point3D &rhs);

  // Display function to print the coordinates
  void display() const {
    cout << "(" << x << ", " << y << ", " << z << ")" << endl;
  }
};

// Overloaded addition operator
Point3D operator+(const Point3D &lhs, const Point3D &rhs) {
  Point3D result;
  result.x = lhs.x + rhs.x;
  result.y = lhs.y + rhs.y;
  result.z = lhs.z + rhs.z;
  return result;
}

// Overloaded subtraction operator
Point3D operator-(const Point3D &lhs, const Point3D &rhs) {
  Point3D result;
  result.x = lhs.x - rhs.x;
  result.y = lhs.y - rhs.y;
  result.z = lhs.z - rhs.z;
```

```
  return result;
}

int main() {
  Point3D p1(7, 8, 9), p2(2, 3, 4);

  cout << "p1: ";
  p1.display();
  cout << "p2: ";
  p2.display();

  cout << "Addition: ";
  Point3D sum = p1 + p2;
  sum.display();

  cout << "Subtraction: ";
  Point3D diff = p1 - p2;
  diff.display();

  return 0;
}
```

**output :**

```
p1: (7, 8, 9)
p2: (2, 3, 4)
Addition: (9, 11, 13)
Subtraction: (5, 5, 5)
```

**3.)**

```
/Write a program to compare two objects of a class that contains an integer
  value as its data member. Make overloading functions to overload
  equality(==), less than(<), greater than(>), not equal (!=), greater than or
  equal to (>=), and less than or equal to(<=) operators using member operator
  functions.*/
#include <iostream>
using namespace std;
class Integer {
private:
  int value;

public:
  Integer(int value = 0) : value(value) {}

  // Overloaded comparison operators
  bool operator==(const Integer &other) const { return value == other.value; }

  bool operator<(const Integer &other) const { return value < other.value; }

  bool operator>(const Integer &other) const { return value > other.value; }
```

```
  bool operator!=(const Integer &other) const { return !(*this == other); }

  bool operator>=(const Integer &other) const {
    return (*this > other) || (*this == other);
  }

  bool operator<=(const Integer &other) const {
    return (*this < other) || (*this == other);
  }

  // Display function to print the value
  void display() const { std::cout << value << std::endl; }
};

int main() {
  Integer a(5), b(10);

  cout << "a: ";
  a.display();
  cout << "b: ";
  b.display();

  cout << "a == b: " << (a == b) << endl;
  cout << "a < b: " << (a < b) << endl;
  cout << "a > b: " << (a > b) << endl;
  cout << "a != b: " << (a != b) << endl;
  cout << "a >= b: " << (a >= b) << endl;
  cout << "a <= b: " << (a <= b) << endl;

  return 0;
}
```

**Output:**

```
a: 5
b: 10
a == b: 0
a < b: 1
a > b: 0
a != b: 1
a >= b: 0
a <= b: 1
```

**4.)**/*Write a class Date that overloads prefix and postfix operators to increase the Date object by one day, while causing appropriate increments to the month and year (use the appropriate condition for leap year). The prefix and postfix operators in the Date class should behave exactly like the built-in increment operators.*/

```
#include <iostream>
```

```
using namespace std;
class Date
{
private:
    int day, month, year;
public:
    Date(int day = 1, int month = 1, int year = 2000) : day(day), month(month),
year(year) {}
    // Prefix increment operator
    Date &operator++()
    {
        // Check if it's the last day of the month
        int lastDay = 31;
        if (month == 2)
        {
            if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
            {
                lastDay = 29;
            }
            else
            {
                lastDay = 28;
            }
        }
        else if (month == 4 || month == 6 || month == 9 || month == 11)
        {
            lastDay = 30;
        }

        if (day == lastDay)
        {
            day = 1;
            if (month == 12)
            {
                month = 1;
                year++;
            }
            else
            {
                month++;
            }
        }
        else
        {
            day++;
        }

        return *this;
```

```
    }

    // Postfix increment operator
    Date operator++(int)
    {
        Date temp(*this);
        ++(*this);
        return temp;
    }

    // Display function to print the date
    void display() const
    {
        cout << day << "/" << month << "/" << year << endl;
    }
};
int main()
{
    Date d(14, 11, 2021);

    cout << "Original date: ";
    d.display();

    cout << "Prefix increment: ";
    (++d).display();

    cout << "Postfix increment: ";
    (d++).display();

    cout << "Final date: ";
    d.display();

    return 0;
}
```

**Output:**

Original date: 14/11/2021
Prefix increment: 15/11/2021
Postfix increment: 15/11/2021
Final date: 16/11/2021