Problema C

Intersección de Vectores de Puntos 3D

autor: RA

En este problema se busca integrar y extender dos clases fundamentales desarrolladas a lo largo del curso: `Punto3D` (que representa un punto en el espacio tridimensional con un color en formato RGB) y `Mivector` (una implementación propia de un arreglo dinámico).

En esta ocasión, la clase `Mivector` ha sido modificada para almacenar objetos de tipo `Punto3D` en lugar de valores de punto flotante. Tu tarea consiste en implementar la sobrecarga del **operador de resta** (`operator-`), el cual calculará la **intersección** entre dos vectores.

Especificación del Operador Resta

El operador debe comportarse de la siguiente manera al calcular la expresión `A - B`:

1. Retorna un nuevo objeto `Mivector` que contiene la intersección de los elementos de `A` y `B`.
2. Un punto se considera común si existe en ambos vectores (utilizando la sobrecarga de `operator==` de la clase `Punto3D`).
3. Los elementos duplicados se preservan en la intersección en la medida en que existan en ambos vectores. Específicamente, si un punto aparece k_A veces en el primer vector (`A`) y k_B veces en el segundo (`B`), el vector resultante debe contener dicho punto exactamente mín(k_A, k_B) veces.
4. Los elementos comunes en el vector resultante deben mantener el **mismo orden de aparición** en el que se encuentran en el vector de la izquierda (`A`).

Clases de Referencia

A continuación se presentan las definiciones de las clases que debes considerar como base:

```
class Punto3D {
private:
    double x, y, z;
    unsigned char color[3]; // RGB color

public:
    Punto3D() : x(0.0), y(0.0), z(0.0) {
        color[0] = color[1] = color[2] = 0;
    }
    Punto3D(double x_arg, double y_arg, double z_arg,
            unsigned char r, unsigned char g, unsigned char b)
        : x(x_arg), y(y_arg), z(z_arg) {
        color[0] = r;
```

```
        color[1] = g;
        color[2] = b;
    }

    double getX() const { return x; }
    double getY() const { return y; }
    double getZ() const { return z; }
    unsigned char getR() const { return color[0]; }
    unsigned char getG() const { return color[1]; }
    unsigned char getB() const { return color[2]; }

    bool operator==(const Punto3D& otro) const {
        return (std::abs(x - otro.x) < 1e-6 &&
                std::abs(y - otro.y) < 1e-6 &&
                std::abs(z - otro.z) < 1e-6 &&
                color[0] == otro.color[0] &&
                color[1] == otro.color[1] &&
                color[2] == otro.color[2]);
    }

    friend std::ostream& operator<<(std::ostream& out, const Punto3D& p) {
        out << "(" << p.x << ",-" << p.y << ",-" << p.z << ")-["
            << (int)p.color[0] << ",-"
            << (int)p.color[1] << ",-"
            << (int)p.color[2] << "]";
        return out;
    }
};

class Mivector {
private:
    Punto3D* datos;
    int tamano;
    int capacidad;

    void redimensionar() {
        capacidad = (capacidad == 0) ? 1 : capacidad * 2;
        Punto3D* nuevos_datos = new Punto3D[capacidad];
        for (int i = 0; i < tamano; ++i) {
            nuevos_datos[i] = datos[i];
        }
        delete[] datos;
        datos = nuevos_datos;
    }

public:
    Mivector() : tamano(0), capacidad(10) {
        datos = new Punto3D[capacidad];
    }
    Mivector(int cap) : tamano(0), capacidad(cap) {
        if (capacidad < 1) capacidad = 1;
        datos = new Punto3D[capacidad];
```

```
    }
    ~Mivector() {
        delete[] datos;
    }
    Mivector(const Mivector& otro)
        : tamano(otro.tamano), capacidad(otro.capacidad) {
        datos = new Punto3D[capacidad];
        for (int i = 0; i < tamano; ++i) datos[i] = otro.datos[i];
    }
    Mivector& operator=(const Mivector& otro) {
        if (this != &otro) {
            delete[] datos;
            tamano = otro.tamano;
            capacidad = otro.capacidad;
            datos = new Punto3D[capacidad];
            for (int i = 0; i < tamano; ++i) datos[i] = otro.datos[i];
        }
        return *this;
    }

    void push_back(Punto3D val) {
        if (tamano == capacidad) redimensionar();
        datos[tamano] = val;
        tamano++;
    }

    int size() const { return tamano; }
    Punto3D& operator[](int index) { return datos[index]; }
    const Punto3D& operator[](int index) const { return datos[index]; }

    // Tarea: Sobrecargar el operador resta para la interseccion
    Mivector operator-(const Mivector& otro) const;

    friend std::ostream& operator<<(std::ostream& out, const Mivector& v) {
        out << "[";
        for (int i = 0; i < v.tamano; ++i) {
            out << v.datos[i] << (i == v.tamano - 1 ? "" : ",");
        }
        out << "]";
        return out;
    }
};
```

Main de Prueba

El código que envíes será probado utilizando un programa principal similar al siguiente:

```
int main() {
    int Na, Nb;
    if (!(cin >> Na >> Nb)) return 0;
```

```
    Mivector A(Na);
    for (int i = 0; i < Na; ++i) {
        double x, y, z;
        int r, g, b;
        cin >> x >> y >> z >> r >> g >> b;
        A.push_back(Punto3D(x, y, z,
                            (unsigned char)r,
                            (unsigned char)g,
                            (unsigned char)b));
    }

    Mivector B(Nb);
    for (int i = 0; i < Nb; ++i) {
        double x, y, z;
        int r, g, b;
        cin >> x >> y >> z >> r >> g >> b;
        B.push_back(Punto3D(x, y, z,
                            (unsigned char)r,
                            (unsigned char)g,
                            (unsigned char)b));
    }

    Mivector C = A - B;
    std::cout << C << std::endl;
    return 0;
}
```

Ejemplos

Ejemplo de entrada	Ejemplo de salida
4 3 1.0 2.0 3.0 255 0 0 4.0 5.0 6.0 0 255 0 7.0 8.0 9.0 0 0 255 1.0 2.0 3.0 255 0 0 7.0 8.0 9.0 0 0 255 10.0 11.0 12.0 255 255 255 1.0 2.0 3.0 255 0 0	[(1, 2, 3) [255, 0, 0], (7, 8, 9) [0, 0, 255]]
3 2 1.0 1.0 1.0 0 0 0 1.0 1.0 1.0 0 0 0 1.0 1.0 1.0 0 0 0 1.0 1.0 1.0 0 0 0 1.0 1.0 1.0 0 0 0	[(1, 1, 1) [0, 0, 0], (1, 1, 1) [0, 0, 0]]