1. Diccionarios Binarios

Un diccionario binario es un árbol binario con orden, es decir, se establece un criterio a través del cual todos los valores de los nodos que están a la izquierda de un nodo son menores al valor de ese nodo y todos los valores de los nodos que están a la derecha de un nodo son mayores o iguales a este.

Se solicita terminar de implementar el siguiente UDT que represente a un diccionario binario.
Probarlo con el main dado.

```
class TreeNode {
public:
    TreeNode(int v) : value(v), left(nullptr), right(nullptr) {};
    ~TreeNode() {
        // ToDo
    }
    void addNode(int v);  // ToDo
    void print();         // ToDo
    unsigned numNodes() const;    // ToDo
    unsigned height() const;      // ToDo
    bool exist(int v);             // ToDo
private:
    int value;
    TreeNode *left;
    TreeNode *right;
};
int main()
{
    static const int NUM_VALUES = 5000;
    static const int MAX_VAL = 10000;
    srand(time(nullptr));
    TreeNode t(rand() % MAX_VAL);
    for (int i = 0; i < NUM_VALUES; ++i)
        t.addNode(rand() % MAX_VAL);
    t.print();
    int n = t.numNodes();
    assert(n == NUM_VALUES);
    cout << "La altura es: " << t.height() << endl;
    int test[] = { 43, 21, 655, 125, 3211, 2244, 4432, 7621, 1234, 4367 };
    for (auto s : test)
        cout << "Existencia de " << s << " en el arbol: " << t.exist(s) << endl;
    return 0;
}
```

2. Matriz rala o matriz dispersa

Una **matriz dispersa** o **matriz rala** es una matriz de gran tamaño en la que la mayoría de sus elementos son cero. Para evitar representar los elementos nulos (la gran mayoría de los elementos) se decide almacenarlos a través de instancias del UDT **`MatrizRala`** que tenga en cuenta esta característica y solo represente los elementos no nulos. Como también se desea tener buena velocidad de acceso a los elementos se decide utilizar como almacenamiento interno un **`std::map`**

en donde se representen los elementos no nulos. La clave de cada entrada será un entero sin signo que representa en forma unívoca al elemento (`i`,`j`) y el valor de cada entrada en el mapa sea el valor no nulo del elemento. Se solicita completar el UDT y definir métodos adicionales si hiciesen falta, como así también la implementación de los operadores de suma y multiplicación.

```
class MatrizRala {
    public:
        MatrizRala(unsigned numFils_, unsigned numCols);           // ToDo

        double getElement(unsigned i, unsigned j) const ;          // ToDo
        double setElement(unsigned i, unsigned j, double value);   // ToDo

        unsigned getNumCols() const;                               // ToDo
        unsigned getNumFils() const;                               // ToDo

    private:
        size_t ij2key(size_t i, size_t j) const {
            // ToDo
            return 0;
        }

        map<size_t, double> elts;
        unsigned numFils;
        unsigned numCols;
};

MatrizRala operator+(const MatrizRala &m1, const MatrizRala &m2);   // ToDo
MatrizRala operator*(const MatrizRala &m1, const MatrizRala &m2);   // ToDo
```

3. Cluster de Estrellas

En astronomía, la distancia entre estrellas cercanas es tan grande que un conjunto de estrellas puede ser modelado como un arreglo de masas puntuales definidas por sus coordenadas y sus masas.
Dado un conjunto de estrellas, se solicita implementar funciones para calcular el centro de masa y el momento de inercia respecto de un eje que pase por el centro de masas.
Las coordenadas del centro de masa se pueden calcular como:

$$CM_x = \frac{\sum_i m_i x_i}{\sum_i m_i} \qquad CM_y = \frac{\sum_i m_i y_i}{\sum_i m_i} \qquad CM_z = \frac{\sum_i m_i z_i}{\sum_i m_i}$$

Donde x_i, y_i, z_i son las coordenadas de la estrella i y m_i su masa.
Por otro lado, el momento de inercia respecto al eje Z que pasa por el centro de masa se define como $I = \sum_i m_i d_i^2$ donde d_i es el módulo del vector bidimensional $[x_i - CM_x,\ y_i - CM_y]$. Termine el diseño e implementación de los siguientes UDT para representar un punto en 3D, una estrella y un cluster de estrellas.

```
class Point3D {
    // ToDo
};

class Star {
    // ToDo
};

class StarCluster {
   public:
      //ToDo
      Point3D centroMasa(); // calcula el centro de masa del cluster
      double inerciaZ(); // calcula el momento de inercia respecto al eje Z
   private:
      // ToDo
};
```