1. Resuelva nuevamente el ejercicio 9 de la práctica 8, utilizando ahora **`std::sort`** para ordenar un **`std::array`** de **`Terna`**s. Pruebe con distintos criterios de ordenamiento: de menor a mayor, de mayor a menor y por diferentes campos.
2. Se desea implementar un functor **`Noiserer`**, que cuando es activado con un **`double`** como argumento retorne un valor que resulte de sumar ese argumento con una componente pseudoaleatoria. La componente pseudo-aleatoria estará limitada entre 2 umbrales **`minV`** y **`maxV`** que serán indicados en la construcción del objeto functor. Verifique su funcionamiento utilizando el functor como función de transformación en la **`std::transform`** desde un **`vector<double> in`** a otro **`vector<doble> out`**, de tal forma que los elementos de **`out`** sean iguales a los de in, pero afectados con un **`Noiserer`**

```
class Noiserer {
    public:
        Noiserer (double minV_, double maxV_);
        double operator()(double v);
    private:
        double minV, maxV;
};
```

3. Dado el siguiente UDT:

```
struct A {
   int val;
   A() {
       val=0;
       cout << "A default: "<<this << '\t' <<  val << endl;
   }
   A(int v) {
       val=v;
       cout<< " A con argumento: "<< this << '\t' <<  val << endl;
   }
   ~A() {
      cout << "~A: "<< this << '\t' << val<<endl;
   }
};
```

Puede decir cuál es la diferencia entre las siguientes sentencias (antes de correr el programa)?:

```
int main(){
   A *pa = new A(10);
   A *pb = new A[10];

// destruccion de pa y pb

   return 0;
}
```

Cómo destruiría **`pa`** y **`pb`**?

4. En una matriz simétrica existe información redundante ya que el elemento (i,j) es igual al elemento (j,i). Se solicita implementar un UDT que permita manejar matrices simétricas evitando esa redundancia. La interface deseada para este UDT es:

```
class SimMatrix {
    public:
        SimMatrix(size_t dim);  // define una matriz de dim x dim
        SimMatrix(const SimMatrix &);  // copy-constructor
        ~SimMatrix();
        void setElement(int i, int j, double val);
        double getElement(int i, int j) const;
        int getDim() const;
    private:
        vector<double> *values;
        size_t dim;
};
```

Implemente los métodos incluyendo el constructor y destructor. Compruebe su correcto funcionamiento.

5. Implemente la función **sumaSimMatrix** que sume matrices simétricas y retorne una nueva **simMatrix** representando el resultado:

```
SimMatrix *sumaSimMatrix(const SimMatrix &m1, const SimMatrix &m2);
```

6. En el ejercicio 4:
   a) piensa que definir un copy-constructor es estrictamente necesario? Justifique.
   b) Si quisiera hacer la operación **a = b**, donde tanto **a** y **b** son instancias de **SimMatrix**, piensa que es necesario redefinir el operador asignación? Justifique y en caso de creerlo necesario, de una implementación.

```
SimMatrix &operator=(const SimMatrix &);
```

7. Con fines didácticos y de aprendizaje se desea implementar el UDT **MyString** para representar strings. El tipo tiene que ser implementado de tal forma de poder soportar el uso siguiente:

```
MyString s1("hola mundo");
MyString s2(s1);
MyString s3;

s3 = s1 + s2;
s3[4] = 'X';
char c = s3[2];
```

8. Se desea representar un polinomio a través de la entidad como la siguiente. Terminar de implementar el UDT.

```
class Polinomio {

    public:
```

```
        // construye un polinomio con los coeficientes y grado dados que
        // representa al polinomio:
        // coefs[0] + coefs[1]*x + coefs[2]*x^2 + ... + coefs[n]*x^n
        Polinomio(const double *coefs, int n);
        Polinomio(const Polinomio &p);
        ~Polinomio();

        Polinomio &operator=(const Polinomio &p);

        Polinomio operator+(const Polinomio &p);
        Polinomio operator*(const Polinomio &p);

        double evaluate(double x); // evalua el polinomio en el valor x

        Polinomio derivate();   // crea y retorna el polinomio derivado

        // crea y retorna el polinomio integrado con constante 'c'
        Polinomio integrate(double c);

    private:
        // puntero a coeficientes
        double *coefs;
        int     grado;
};
```

9. Una empresa que realiza liquidaciones de sueldo en efectivo, requiere un sistema para el cálculo de las cantidades de billetes / monedas de las diferentes denominaciones que hacen falta para pagar una cantidad dada de sueldos. Estas cantidades serán luego solicitadas al banco.
   El sistema planteado, define las siguientes entidades:

```
class Monedero {
    public:
        // construye un monedero con las distintas denominaciones
        // a manejar expresadas en centavos. Por ejemplo si se van a manejar
        // las siguientes denominaciones: $0.50, $1, $2, $10, $50, $100 y $500
        // podría construirse con el vector
        // {50, 100, 200, 2000, 5000, 10000, 50000
        Monedero(const vector<int> &valorMonedas);

        // Agrega un sueldo a ser distribuido en las distintas
        // denominaciones, se debe minimizar la cantidad de
        // billetes/monedas a entregar
        void agregaSueldo(double montoSueldo);

        // Retorna la cantidad de billetes/monedas necesarios de la
        // denominacion de valor 'denominacion'
        int cantidadValor(int denominacion);

    private:
        struct Denominacion {
            int valor;    // valor de la denominación en centavos
            int cantidad; // cantidad de billetes/moneda necesarios
        };
        vector<Denominacion> monedas; // denominaciones a utilizar
};
```

   Se solicita terminar de implementar el UDT. Evalúe la necesidad de implementar un constructor de copia, un operador de asignación y un destructor. En el caso de llegar a la conclusión de que hacen falta, impleméntelos.