1. Implemente la siguientes funciones:

```
// Alloca y devuelve un vector de n doubles
double *AllocVector(int n);

// libera el vector
void FreeVector(double *pVect);

// completa el vector con valores aleatorios en [0,1]
void FillRandVector(double *pVect, int n);

// Ordena el vector en forma creciente utilizando qsort
void SortVector(double *pVect, int n);

// imprime el vector
void PrintVector(double *pVect, int n);
```

   Realice un programa de prueba que permita medir el tiempo que lleva ordenar un vector utilizando **`SortVector`**, y ejecútelo para distintos valores de **`n`**.

2. En una matriz simétrica existe información redundante ya que el elemento (i,j) es igual al elemento (j,i). Por eso para almacenar todos los datos es necesario contar con 1/2 *N*(N+1) (< N*N) lugares de memoria.
   Implemente un conjunto de funciones que permita crear y acceder a elementos individuales de matrices simétricas:

```
double **AllocaMatSimetrica(int n);
void LiberaMatSimetrica(double **pMat, int n);
double GetElement(double **pMat, int n, int i, int j);
void SetElement(double **pMat, int n, int i, int j, double X);
```

3. Utilizando las funciones anteriores, defina los prototipos e implemente las funciones que realicen la suma y multiplicación de 2 matrices simétricas. Estas funciones deben retornar una nueva matriz resultado (que no es simétrica).

4. Implemente las funciones faltantes para el manejo de las estructuras Polinomio vista en la teórica:

```
double PolEval(Polinomio_t *pPol, double x);

Polinomio_t *PolDeriv(Polinomio_t *pPol);

Polinomio_t *PolInteg(Polinomio_t *pPol);
```

5. Dadas las siguientes estructuras de datos:

```
typedef struct {
    double X,
           Y;
} Punto2D;

typedef struct {
    int      numPuntos;
    Punto2D *Vertices;
} Poligono;
```

Implementar las siguientes funciones:

- Función que crea (alloca) la estructura de datos para mantener un polígono de n vértices. Debe devolver un puntero a la estructura creada, o NULL si ocurrió algún error.

  ```
  Poligono *creaPoligono(int N);
  ```

- Función que impone valores al n-ésimo vértice del polígono que recibe como argumento. Esta función debe retornar 0 si todo fue OK o distinto de 0 si ocurrió algún error.

  ```
  int setVertice(Poligono *p, int n, double x, double y);
  ```

- Función que calcula y devuelve el perímetro del polígono que recibe como argumento.

  ```
  double perimetro(Poligono *p);
  ```

- Función que calcula y devuelve el área del polígono que recibe como argumento.

  ```
  double area(Poligono *p);
  ```