Importante: Los ejercicios deben entregarse a través de web (**Domjudge y Blackboard**). En un fichero con nombre:

polinomio.cpp (Utiliza como base el fichero proporcionado en Blackboard)

main.cpp (Utiliza como base el fichero proporcionado en Blackboard)

SolucionParcial.cpp (Utiliza como base el fichero proporcionado en Blackboard)

La fecha de entrega: consultar la página de la actividad en blackboard

Raíces de un polinomio mediante algoritmos genéticos. Implementar un algoritmo genético que permita calcular las raíces de un polinomio. Para ello deberá crearse la clase **Polinomio**:

- Un atributo con el grado del polinomio.
- Un atributo que será un puntero a un array con los coeficientes del polinomio (datos de tipo float). El array tendrá que tener espacio para n+1 componentes. En cada celda del array guardaremos el coeficiente del término del grado i, siendo i la posición de la celda. Ej: el polinomio $p(x)=8x^5 +5x - 3.2$ se guardará en el siguiente array: [-3.2, 5, 0, 0, 0, 8]
- Un **constructor**, al que se le pasa el grado del polinomio y la dirección de comienzo del array de coeficientes. El constructor copiará el contenido de ese array a su propio array interno de coeficientes.
- Método público **evaluar** que recibe el valor de x (float) y nos devuelve el valor del polinomio para esa x (otro float).
- Método público **obtenerRaizRecursivo** que será el que implemente el algoritmo genético recursivo para obtener la raíz del polinomio. Este método utilizará la clase SolucionParcial que contenga el "x" y el "y" de una solución parcial. Este método recibirá una solución parcial como parámetro y devolverá un valor de x como solución.
- Este método funcionará de acuerdo a las siguientes características:
  - La solución parcial inicial será x=0.
  - En cada generación se crearán 10 hijos con mutaciones del padre.
  - La mutación de cada hijo significa la suma de la mutación en "x" del hijo con la "x" del padre. Por ejemplo, si el padre tiene una x=5, y el hijo tiene una mutación de -0.2, significa que el hijo tiene una x=4.8
  - Por cada generación sobrevive únicamente el mejor hijo.
  - Para simplificar, pararemos el algoritmo cuando, en una generación, todos los hijos sean peores que el padre. El resultado final será entonces el x de ese padre.
  - Esta función se puede hacer recursiva o iterativa. Es más sencilla recursiva. En cada iteración, se imprimirá por pantalla el "x" y "p(x)" del padre usando el método de imprimir de SolucionParcial.

NOTA: Las mutaciones deberán seguir una distribución de probabilidad “normal” y “estándar”(es decir, con media 0 y desviación típica 1). Por lo tanto, no puedes usar un simple rand(), pues su distribución de probabilidad es homogénea, no normal estándar. Un número aleatorio normal estándar se puede calcular o bien con "normal_distribution" de la biblioteca "random" (versión 2011 de C++), o bien mediante 12 sumas sucesivas de valores aleatorios uniformes entre 0 y 1 (rand()/(float)RAND_MAX), y luego restándole 6. Más detalles sobre este último método
http://en.wikipedia.org/wiki/Normal_distribution#Generating_values_from_normal_distribution.

El **input** constará de tres partes. El primer número será la semilla aleatoria usada para inicializar rand para que de esta forma todas las implementaciones obtengan los mismos valores aleatorios. El segundo número será n el número de coeficientes del polinomio del que calcular la raíz. El tercero será una lista de n números con los coeficientes del polinomio.

El **output** será las líneas por cada iteración del algoritmo de genético como se indica en la implementación.

Input:	Output:
1 2 6 -5 1	Seleccionada (0,6) Mutaciones (0.966774,2.10078) (0.0487857,5.75845) (-0.56404,9.13834) (1.48117,0.788007) (-1.66931,17.1331) (0.65507,3.15377) (1.21211,1.40867) (1.74353,0.322251) (1.47885,0.792754) (-0.81962,10.7699) Seleccionada

	(1.74353,0.322251) Mutaciones (1.86094,0.158399) (0.883812,2.36206) (1.54024,0.671138) (1.05758,1.83058) (1.72729,0.347079) (2.51335,-0.249822) (4.9766,5.88356) (1.64144,0.487117) (1.0266,1.9209) (1.83513,0.192058) Seleccionada (1.86094,0.158399) Mutaciones (1.78513,0.261038) (2.14676,-0.125224) (0.996882,2.00936) (2.09356,-0.0848026) (3.55641,0.865999) (3.02737,0.0281248) (1.43727,0.879403) (2.25269,-0.188836) (-0.164881,6.85159) (2.37128,-0.233431) Seleccionada (3.02737,0.0281248) Mutaciones

	(3.53054,0.812011) (3.18877,0.224404) (1.15832,1.55011) (3.36785,0.503161) (2.35015,-0.227544) (2.97803,-0.0214901) (2.71667,-0.203052) (1.45653,0.83883) (3.94372,1.83433) (2.70352,-0.20858) Seleccionada (2.97803,-0.0214901) Mutaciones (1.66182,0.45255) (3.82268,1.49949) (4.85791,5.30976) (5.89889,11.3025) (4.27552,2.90246) (2.68859,-0.214435) (1.77158,0.2806) (2.68007,-0.217576) (5.45971,8.50986) (4.29314,2.96535) Iteracion 0 Raiz 2.97803 ….

NOTA: Para cada una de las funciones implementados se deberá incluir una pequeña descripción de su funcionamiento, sus precondiciones mediante assertdomjudge si las hubiera y el análisis de su complejidad temporal y espacial. Esta información deberá incluirse en la cabecera de cada función.