Práctica 6

Importante: Los ficheros deben entregarse a través de web (por **Domjudge** y **Blackboard**).

Para los ejercicio se deberán entregar 3 **ficheros** con nombres:

polinomio.h, polinomio.cpp, solucionparcial.h y main.cpp

Estos ficheros serán proporcionados a través de Blackboard y deberán ser modificados.

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

En esta hoja se pretende desarrollar la clase "Polinomio" que representa un polinomio y que además permite realizar operaciones sobre ellos.

La clase Polinomio tiene los siguientes atributos:

- **int n**: atributo que indica el grado del polinomio.
- **float * coeficientes**: 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)=8x5+5x-3.2 se guardará en el siguiente array: [-3.2, 5, 0, 0, 0, 8]
- **const int numeroHijos**: constante con valor 10 que indica el número de hijos que se deberán crear en cada iteración del algoritmo genético

La clase SolucionParcial tiene los siguientes atributos y métodos.

- f**loat x:** atributo que almacena el valor de x de la solución parcial actual
- float y: atributo que almacena el resultado de evaluar x sobre el polinomio p(x)
- imprimir(): método que imprime por pantalla el par (x,p(x))

Para la clase Polinomio deberán implementarse los siguientes métodos públicos:

- **(0,5 puntos) Constructor por parámetros. Polinomio(int n, float * coefs)** 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. Además deberá inicializar la variable que almacena el número de hijos del algoritmo genético.
- **(0,5 puntos)Destructor.** Se encargará de liberar la memoria que fue reservada de forma dinámica para almacenar el polinomio.
- **(1 puntos) float evaluar(float x)**", método público que recibe el valor de x (float) y nos devuelve el valor del polinomio para esa x (otro float).
- **float obtenerAleatorioNormalEstandar().** Este método público está implementado y se encarga de generar número aleatorios con una distribución de probabilidad normal estándar (con media 0 y desviación típica 1). El cálculo se realiza mediante 12 sumas sucesivas de valores aleatorios uniformes entre 0 y 1 (nos lo da rand()/(float)RAND_MAX), y luego restándole el número 6.

Más detalles sobre este método en el apartado "Generating values from normal distribution" de la web:

http://en.wikipedia.org/wiki/Normal_distribution.

Para calcular una normal no estándar (es decir con media y desviación distintas de 0 y 1 respectivamente) recordemos que se calcularía: media + desviación*Z, donde Z es el aleatorio calculado mediante la distribución normal estándar.

- (1 puntos**) float obtenerRaiz ().** Método público que implementará el algoritmo genético para obtener las raíces del polinomio. No tendrá ningún parámetro, y devolverá el valor de x como solución. Internamente llamará a una función privada recursiva **obtenerRaíz Recursiva**. La solución parcial inicial será x=0.
- **(4 puntos) float obtenerRaizRecursiva(SolucionParcial parcial).** Método privado que implementa la llamada recursiva del algoritmo genético sobre una solución parcial para aproximarse a la solución.
  - Este algoritmo deberá crear tantos hijos como indique **numeroHijos** con mutaciones en sus valores x mediante la función **obtenerNumeroAleatorioStandard()**.
  - La mutación de cada hijo significa sumar la mutación al "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
  - Para simplificar, por cada generación sobrevive únicamente el mejor hijo es decir cuyo valor de p(x) es más cercano a 0.
  - 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.
  - En cada iteración, se imprimirá por pantalla el "x" y "p(x)" del padre (en formato "(x, p(x))") mediante le método imprimir de SolucionParcial y a continuación de cada uno de los hijos.
    OJO: viola el principio de separación entre interfaz y modelo… pero igualmente lo haremos sólo por esta vez pues, en caso contrario, programar esta traza se complicaría excesivamente en este ejercicio.

La entrada del programa principal tendrá el siguiente formato

- "%d" Indicando la semilla aleatoria utilizada para generar los números aleatorios
- "%d" Grado del polinomio del que se va a calcular la raíz
- "%f ..%f" Los coeficientes del polinomio. Habran grado+1 coeficientes

La salida del programa principal será la generada por obteneRaizRecursiva y las raíces obtenidas para cada una de las 5 iteraciones.

NOTA: Para cada uno de los métodos 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.

A continuación se puede observar un ejemplo de Entrada y Salida

Entrada:

```
1
2
6 -5  1
```

Salida:

```
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

Seleccionada
(0,6)
Mutaciones
(0.796237,2.65281)
(-0.503158,8.76896)
(-0.0846109,6.43021)
(0.979962,2.06051)
(-0.160098,6.82612)
(-1.59292,16.502)
(-0.432881,8.35179)
(-0.268585,7.41506)
(0.861189,2.4357)
(-1.17147,13.2297)

```
Seleccionada
(0.979962,2.06051)
Mutaciones
(1.49953,0.750941)
(0.287747,4.64406)
(1.23537,1.34929)
(1.99814,0.00186419)
(-0.881594,11.1852)
(0.361221,4.32437)
(1.91101,0.0969062)
(1.21785,1.39391)
(-0.130546,6.66977)
(0.664686,3.11838)
Seleccionada
(1.99814,0.00186419)
Mutaciones
(0.0716491,5.64689)
(2.08182,-0.0751219)
(2.46912,-0.249046)
(4.12726,2.39797)
(1.74168,0.325043)
(0.978406,2.06525)
(3.41037,0.578769)
(1.49862,0.752752)
(1.79611,0.245466)
(1.67141,0.436556)
Iteracion 1 Raiz 1.99814

Seleccionada
(0,6)
Mutaciones
(-1.37201,14.7424)
(0.268132,4.73124)
(0.895543,2.32428)
(-0.840726,10.9104)
(-1.16618,13.1909)
(0.499277,3.75289)
(-0.264801,7.39412)
(-0.504167,8.77502)
(0.483928,3.81455)
(1.09241,1.73132)
Seleccionada
(1.09241,1.73132)
Mutaciones
```

(0.563601,3.49964)
(2.20696,-0.16413)
(0.728547,2.88805)
(1.23312,1.35499)
(1.21974,1.38906)
(0.987803,2.03674)
(1.55022,0.652073)
(-1.60065,16.5654)
(1.51533,0.719578)
(0.665383,3.11582)
Seleccionada
(2.20696,-0.16413)
Mutaciones
(4.256,2.83354)
(2.8396,-0.134669)
(1.57265,0.609974)
(1.57512,0.605402)
(2.9274,-0.0673332)
(2.81699,-0.149514)
(0.864357,2.42533)
(0.912976,2.26865)
(2.18471,-0.150596)
(2.30757,-0.212971)
Seleccionada
(2.9274,-0.0673332)
Mutaciones
(1.59138,0.575595)
(3.61019,0.982522)
(1.70554,0.381173)
(2.05502,-0.0519938)
(2.90854,-0.0830994)
(3.37344,0.512901)
(1.71146,0.371799)
(3.49897,0.747936)
(3.27417,0.34934)
(2.71906,-0.202011)
Seleccionada
(2.05502,-0.0519938)
Mutaciones
(2.24863,-0.186814)
(1.94163,0.0617733)
(0.682629,3.05284)
(2.12136,-0.106628)
(1.24171,1.33328)

(2.82795,-0.142451)
(2.65538,-0.225858)
(2.586,-0.242603)
(2.2715,-0.197787)
(2.51416,-0.249799)
Iteracion 2 Raiz 2.05502

Seleccionada
(0,6)
Mutaciones
(-0.343791,7.83715)
(0.653796,3.15847)
(-1.91532,19.2451)
(0.775802,2.72286)
(-0.171068,6.8846)
(-0.69304,9.9455)
(0.360559,4.32721)
(1.74534,0.319513)
(0.362469,4.31904)
(-0.0703273,6.35658)
Seleccionada
(1.74534,0.319513)
Mutaciones
(2.24957,-0.187284)
(1.70337,0.384612)
(1.25754,1.2937)
(1.10503,1.69593)
(1.21606,1.39851)
(3.12156,0.136337)
(4.41473,3.41619)
(3.00426,0.00427628)
(-0.284108,7.50126)
(1.64009,0.48944)
Seleccionada
(3.00426,0.00427628)
Mutaciones
(2.16476,-0.137613)
(3.58868,0.935233)
(2.91112,-0.0809784)
(1.94726,0.0555174)
(3.96041,1.88279)
(3.77975,1.38776)
(1.72049,0.357633)
(4.12975,2.40608)

(2.9927,-0.00724792)
(0.332179,4.44945)
Iteracion 3 Raiz 3.00426

Seleccionada
(0,6)
Mutaciones
(1.65878,0.457648)
(-0.660493,9.73872)
(0.0148239,5.9261)
(-1.29006,14.1145)
(1.29757,1.19584)
(0.406535,4.1326)
(-2.30411,22.8295)
(0.277351,4.69017)
(-1.3209,14.3493)
(-0.0731721,6.37121)
Seleccionada
(1.65878,0.457648)
Mutaciones
(2.36949,-0.232967)
(1.29041,1.2131)
(1.06204,1.81773)
(2.30983,-0.213836)
(3.18984,0.225885)
(0.960541,2.11993)
(-0.924735,11.4788)
(2.57105,-0.244951)
(1.85637,0.164255)
(2.19521,-0.157103)
Seleccionada
(2.19521,-0.157103)
Mutaciones
(0.216399,4.96483)
(1.79442,0.247848)
(2.80253,-0.158475)
(3.0304,0.0313263)
(1.95373,0.0484071)
(3.30194,0.393114)
(2.9756,-0.0238085)
(1.41663,0.92369)
(3.42597,0.607418)
(0.859604,2.4409)
Seleccionada

(2.9756,-0.0238085)
Mutaciones
(4.7184,4.67129)
(3.914,1.74939)
(2.21962,-0.171386)
(3.51892,0.788198)
(3.43261,0.619766)
(1.42013,0.916109)
(4.60275,4.17157)
(1.36633,1.0352)
(2.21565,-0.169143)
(3.1978,0.236923)
Iteracion 4 Raiz 2.9756