



**Instituto
Politécnico
Nacional**



REPORTE DE PRACTICA P11

POLINOMIO CON COEFICIENTES RACIONALES

Julio Cesar Hernández Reyes

**Grupo:
2CV11**

**Profesor:
Juan Jesús Gutiérrez García**

**Algoritmos y Estructura de Datos
Sep-2020**

RESPONDIENDO A LO QUE PIDIÓ EN MOODLE

a.- ¿Por qué elegí esta opción?

Elegí la opción 1 porque tenía tiempo para pensar y llegar a una solución para este problema además de que creo tengo un poco de nivel en la programación, no tanto pero si lo suficiente para poder realizar este proyecto.

b.- ¿Cómo la resolví?

El proyecto lo empecé haciendo un código main aparte que no utilizara ninguna función de las ya proporcionadas, que resolviera y encontrara las raíces de un polinomio que ingreses manualmente dando los valores de a, de b y de c. Este código me dio la idea en general de que se tenía que hacer para el main principal que debía probar la función que solucionara esto mismo, pero para racionales y con racionales. Me costo entender cómo funcionaba el código proporcionado, pero no fue nada que no pudiera realizar. Con el paso de los días me di cuenta de una cosa tras otra hasta que complete el proyecto creo yo para los diferentes casos que pude solucionar como es el caso de que solo haya una solución o dos soluciones, además de los casos en que a es 0. Lo que le falta a mi código es que la función raíz saque la raíz de un racional de forma mas exacta y que se represente en racionales, como no pude hacer esto el código funciona, pero en unos casos por no decir la mayoría en la comprobación no da 0 exacto, solo llegue a aproximarme al cero, pero no siempre se obtiene pues como mencione antes la raíz que se obtiene no es del todo exacta.

c.- ¿Qué pruebas hice para asegurar que mi respuesta es correcta?

Las pruebas que hice fue crear un archivo main en el que se encuentran varios ciclos en los que dependiendo el valor en el que vayan las variables proporcionadas estos se guardaban como el valor de los racionales que componían a una ecuación del tipo polinomial de segundo grado.

Existen 3 ciclos anidados con diferentes variaciones haciendo que los números sean diferentes en cada uno. Cabe mencionar que mi código solo imprime los polinomios y las soluciones de los que si tienen solución pues los que no simplemente no se imprimen en consola.

Trabaje en Windows utilizando gcc gracias al instalador MWInstaler que nos proporciono y me funciono de maravilla pues anteriormente trabajaba en una maquina virtual y se me hacía más difícil trabajar de esa forma.

DESCRIPCIÓN DEL PROBLEMA

El objetivo de la práctica es construir un tipo de dato que represente a los polinomios de segundo grado con coeficientes racionales. Un ejemplo de estos puede ser:

$$\frac{3}{4}x^2 + \frac{3}{22}x + \frac{1}{3} = 0$$

En este caso se trata de un polinomio de segundo grado con coeficientes racionales, el objetivo es obtener las raíces del polinomio que esté expresada como una fracción.

Para hacerlo se crearon los siguientes tipos de datos

- Racional, que representa al numerador, denominador y signo de un número racional. Este tipo está definido en racional.h e implementado en racional.c
- Ecseg, es el tipo que representa una ecuación de segundo grado con coeficientes racionales. Además, mantiene dos campos o atributos racionales x1 y x2 que almacenarán la solución de la ecuación. Y un entero que es el estado en que se encuentra el número racional:

IMPLEMENTACIÓN DEL RACIONAL

Para implementar el número racional se parte de la siguiente estructura que está definida en el archivo racional.h

```

/*****- Definición del tipo racional-*****/
typedef struct{
    int num; /*númerador, siempre positivo*/
    int den; /*denominador, siempre positivo */
    int sig; /* 0 negativo 1 positivo*/
} racional;

```

Tanto el denominador como el numerador siempre serán números positivos, habría que evaluar declararlos como unsigned int. Así el signo del número completo estará en el entero sig.

La función que inicializa el número racional recibirá parámetros enteros con signo y los convertirá a positivos para almacenarlos en la estructura y calculará el valor de sig. Como la fracción siempre se almacena reducida, es decir el $\frac{4}{2}$ se guarda como $\frac{1}{2}$, es necesario calcular el máximo común divisor. Hay dos funciones que se usan para ello uno que verifica los parámetros y la otra que hace el proceso recursivo del cálculo.

Las funciones implementadas son:

- a) inic_rac, que le da valores iniciales a numerador y denominador, esta función verifica que el denominador no sea cero.
- b) tostr_rac, convierte un número racional en una representación en cadena para poder ser mostrada con la función printf.
- c) copia_rac, realiza la copia de un racional en otro.
- d) comp_rac, compara dos racionales, si el primer parámetro igual con el segundo parámetro.
- e) dist_rac, dice si dos racionales son diferentes
- f) mayq_rac, dice si el primer parámetro es mayor que el segundo
- g) inv_rac, invierte el número racional.
- h) raíz_rac, calcula la raíz cuadrada de un número racional
- i) suma_rac, calcula la suma entre dos racionales.
- j) rest_rac, calcula la resta entre dos racionales.
- k) mult_rac, calcula la multiplicación de dos racionales.
- l) divi_rac, calcula la división entre dos racionales
- m) mcd_r, Realiza el cálculo recursivo del máximo común divisor

En el caso de representar el polinomio de segundo grado con coeficientes racionales utilice la siguiente estructura:

```

/****- Definición del tipo ecuación de segundo grado-*****/
typedef struct{
    /*Coeficientes*/
    racional a; /*Variable al cuadrado*/
    racional b; /*Variable lineal */
    racional c; /*Constante*/
    /*Raíces de la ecuación*/
    racional x1;
    racional x2;
    /*Estado*/
    int estado; /*Entero que define el estado de la ecuación*/
} ecseg;

```

Las funciones utilizadas para este tipo son:

- a) ini_ec2r, Inicializa una ecuación como polinomio de segundo grado

b) `tost_ec2r`, Convierte una ecuación a cadena

c) `sol_ecseg`, Obtiene la solución para un polinomio de segundo grado

Partes que se añadieron al código proporcionado:

Las inicializaciones de las funciones multiplicación y división de racionales:

```
/*Multiplicacion de dos números racionales
Recibe: Tres parámetros
    Apuntador a un racional que almacenará el resultado
    Apuntador a el primer racional para multiplpicar
    Apuntador a el segundo racional para multiplpicar
Regresa:
    APINV: en caso de recibir un apuntador invalido
    OK: Si no hay error
*/
int mult_rac(racional*, const racional*, const racional*);

/*Division de dos números racionales
Recibe: Tres parámetros
    Apuntador a un racional que almacenará el resultado
    Apuntador a el primer racional para ser el dividendo
    Apuntador a el segundo racional para ser el divisor
Regresa:
    APINV: en caso de recibir un apuntador invalido
    OK: Si no hay error
*/
int divi_rac(racional*, const racional*, const racional*);
```

Ya las funciones desarrolladas en el archivo `racional.c`

```
/*Multiplica dos números racionales
Recibe: Tres parámetros
    Apuntador a un racional que almacenará el resultado
    Apuntador a el primer racional para multiplicarse
    Apuntador a el segundo racional para multiplicarse
Regresa:
    APINV: en caso de recibir un apuntador invalido
    OK: Si no hay error
*/
int mult_rac(racional* r, const racional* a, const racional* b){
    int num,den,err;
    /* Valida */
    if(!(APVAL(r)&&APVAL(a)&&APVAL(b)))
        return APINV;
    num = (SIGNOV(a)*a->num) * (SIGNOV(b)*b->num);
    den = (a->den)*(b->den);
    err = ini_rac(r,num,den);
    return err;
}
```

```

/*Division dos números racionales
Recibe: Tres parámetros
    Apuntador a un racional que almacenará el resultado
    Apuntador a el primer racional para ser el dividendo
    Apuntador a el segundo racional para ser el divisor
Regresa:
    APINV: en caso de recibir un apuntador invalido
    OK: Si no hay error
*/
int divi_rac(racional* r, const racional* a, const racional* b){
    int num,den,err;
    /* Valida */
    if(!(APVAL(r)&&APVAL(a)&&APVAL(b)))
        return APINV;

    num= (SIGNOV(a)*a->num) * (SIGNOV(b)*b->den);
    den= a->den * b->num;
    err = ini_rac(r,num,den);
    return err;
}

```

Inicialización de la función que solucionara los polinomios de segundo grado en pol2rac.h:

```

/* Saca las raices del polinomio ingresado pero como no pude
reducir mucho las raices hay uno que otro fallo en los resultados
pues de repente en unps casos se aleja mucho del cero.
*/
void sol_ecseg(ecseg* e, const racional* a, const racional* b, const racional* c);

```

Aquí empieza lo que implemente de la función para solucionar los polinomios de segundo grado:

```

72  /* Esta función es la que se encarga de obtener las raices de los polinomios proporcionados por el main
73     Dependiendo de las condiciones de cada polinomio se hacen diferentes cosas para obtener las raices
74     En esta función solo saca las raices cuando a es 0, cuando la raíz de la fórmula es mayor a cero,
75     y cuando la raíz de la fórmula es igual a cero.
76     No se tomo en cuenta la situación en la que la raíz es negativa pues el resultado seria soluciones imaginarias.
77  */
78
79  void sol_ecseg(ecseg* e, const racional* a, const racional* b, const racional* c){
80      //declaración de todas las variables racionales que se usaran en el calculo de las raices
81      racional D,p,q,r,m1,m2,m3,m4,m5,m6,div1,div2,div3,div4,raiz1,suma1,rest1,rest2,minusone,two,four,x1,x2;
82      //se igualan los numeros racionales dados por p,q y r para poder trabajar con ellos en esta función
83      p=*a;
84      q=*b;
85      r=*c;
86      //se declaran estos enteros para despues hacer comparaciones
87      //entre racionales y dependiendo que se hace u na cosa u otra
88      int dif,dif1,azero;
89      char s1[1000]; //variable para guardar los valores y luego convertirlos a cadena.
90      racional cero; //racional cero para hacer comparaciones de racionales
91      ini_rac(&cero,0,1);
92      ini_rac(&minusone,-1,1); //racional -1 que sera el que hara negativos a los racionales que se necesiten
93      ini_rac(&two,2,1); //valor de 2a en la ecuación de las raices de los polinomios
94      ini_rac(&four,4,1); //valor de 4b en la ecuación de las raices de los polinomios

```

```

95 //Obtencion de D que seria lo que esta dentro de la raiz de la formula
96 mult_rac(&m1,&q,&q);
97 mult_rac(&m2,&p,&r);
98 mult_rac(&m3,&four,&m2);
99 rest_rac(&rest1,&m1,&m3);
100 D=rest1;// resultado de D --> -b+4ac
101 dif=comp_rac(&D,&cero);//comparacion de lo que esta en la raiz y el racional cero 1-->iguales 0--diferentes
102 dif1=mayq_rac(&D,&cero);//compara si lo que esta en la raiz es mayor a cero
103 azero=comp_rac(&p,&cero);//verifica si el valor a del polinomio es cero
104 if(azero==0){//lo que se hace si el valor a del polinomio no es 0
105 if(dif==1){//lo que se hace si D es igual a cero
106 //impresion de la formula
107 printf("\n\n%sx^2",tostr_rac(s1,&p));
108 printf("+%sx",tostr_rac(s1,&q));
109 printf("+%s=0\n",tostr_rac(s1,&r));
110
111 printf("El polinomio tiene una sola solucion:\n");
112 mult_rac(&m5,&minusone,&q);//calcula de -b
113 mult_rac(&m4,&two,&p);//calcula de 2a
114 divi_rac(&div1,&m5,&m4);//Calcula de -b/2a
115 x1=div1;//igualacion del resultado a x1
116 printf("%s\n",tostr_rac(s1,&div1));//impresion de la raiz
117
118 //comprobacion;
119 printf("Comprobacion de soluciones:\n");
120 racional ax_2,x_2,bx,cx,sumaparte,sumacompleta;
121 mult_rac(&x_2,&x1,&x1);
122 mult_rac(&ax_2,&p,&x_2);
123 mult_rac(&bx,&q,&x1);
124 mult_rac(&cx,&r,&x1);
125 suma_rac(&sumaparte,&ax_2,&bx);
126 suma_rac(&sumacompleta,&sumaparte,&cx);
127
128 printf("siendo x=%s\n",tostr_rac(s1,&x1));
129 printf("%s",tostr_rac(s1,&p));
130 printf("%s^2",tostr_rac(s1,&x1));
131 printf("+%s",tostr_rac(s1,&q));
132 printf("%s",tostr_rac(s1,&x1));
133 printf("+%s",tostr_rac(s1,&r));
134 printf("%s=0\n",tostr_rac(s1,&x1));
135
136 printf("%s",tostr_rac(s1,&p));
137 printf("%s",tostr_rac(s1,&x_2));
138 printf("+%s",tostr_rac(s1,&bx));
139 printf("+%s=0\n",tostr_rac(s1,&cx));
140
141 printf("%s",tostr_rac(s1,&ax_2));
142 printf("+%s",tostr_rac(s1,&bx));
143 printf("+%s=0\n",tostr_rac(s1,&cx));
144
145 printf("%s",tostr_rac(s1,&sumaparte));
146 printf("+%s=0\n",tostr_rac(s1,&cx));
147
148 printf("%s=0\n",tostr_rac(s1,&sumacompleta));
149 //Se guarda el valor de x1 en la ecuacion e
150 e->x1=x1;
151

```



```

152
153 }else if(dif1==1){//lo que se hace si D es mayor a cero
154     //Impresion de la formula
155     printf("\n\n%sx^2",tostr_rac(s1,&p));
156     printf("+%sx",tostr_rac(s1,&q));
157     printf("+%s=0\n",tostr_rac(s1,&r));
158
159     printf("El polinomio tiene dos soluciones:\n");
160     mult_rac(&m5,&minusone,&q);//-b
161     raiz_rac(&raiz1,&D);//raiz de D
162     suma_rac(&suma1,&m5,&raiz1);//-b+Raiz(D)
163     mult_rac(&m4,&two,&p);//calculo de 2a
164     divi_rac(&divi2,&suma1,&m4);//calculo de (-b+Raiz(D))/2a
165     x1=divi2;//igualacion del resultado a x1
166     printf("Solucion 1:%s\n",tostr_rac(s1,&x1));//impresion de la raiz 1
167
168     rest_rac(&rest2,&m5,&raiz1);//-b+Raiz(D)
169     divi_rac(&divi3,&rest2,&m4);//calculo de (-b-Raiz(D))/2a
170     x2=divi3;//igualacion del resultado de x2
171     printf("Solucion 2:%s\n",tostr_rac(s1,&x2));
172     //comprobacion de soluciones
173     printf("Comprobacion de soluciones:\n");
174     racional ax_2,x_2,bx,cx,sumaparte,sumacompleta;
175     mult_rac(&x_2,&x1,&x1);
176     mult_rac(&ax_2,&p,&x_2);
177     mult_rac(&bx,&q,&x1);
178     mult_rac(&cx,&r,&x1);
179     suma_rac(&sumaparte,&ax_2,&bx);
180     suma_rac(&sumacompleta,&sumaparte,&cx);
181
182     //comprobacion para la raiz 1
183     printf("siendo x1=%s\n",tostr_rac(s1,&x1));
184     printf("%s",tostr_rac(s1,&p));
185     printf("%s^2",tostr_rac(s1,&x1));
186     printf("+%s",tostr_rac(s1,&q));
187     printf("%s",tostr_rac(s1,&x1));
188     printf("+%s",tostr_rac(s1,&r));
189     printf("%s=0\n",tostr_rac(s1,&x1));
190

```

```

191     printf("%s",tostr_rac(s1,&p));
192     printf("%s",tostr_rac(s1,&x_2));
193     printf("+%s",tostr_rac(s1,&bx));
194     printf("+%s=0\n",tostr_rac(s1,&cx));
195
196     printf("%s",tostr_rac(s1,&ax_2));
197     printf("+%s",tostr_rac(s1,&bx));
198     printf("+%s=0\n",tostr_rac(s1,&cx));
199
200     printf("%s",tostr_rac(s1,&sumaparte));
201     printf("+%s=0\n",tostr_rac(s1,&cx));
202
203     printf("%s=0\n",tostr_rac(s1,&sumacompleta));
204
205     //comprobacion para la raiz 2
206     printf("siendo x2=%s\n",tostr_rac(s1,&x2));
207     printf("%s",tostr_rac(s1,&p));
208     printf("%s^2",tostr_rac(s1,&x2));
209     printf("+%s",tostr_rac(s1,&q));
210     printf("%s",tostr_rac(s1,&x2));
211     printf("+%s",tostr_rac(s1,&r));
212     printf("%s=0\n",tostr_rac(s1,&x2));
213
214     printf("%s",tostr_rac(s1,&p));
215     printf("%s",tostr_rac(s1,&x_2));
216     printf("+%s",tostr_rac(s1,&bx));
217     printf("+%s=0\n",tostr_rac(s1,&cx));
218
219     printf("%s",tostr_rac(s1,&ax_2));
220     printf("+%s",tostr_rac(s1,&bx));
221     printf("+%s=0\n",tostr_rac(s1,&cx));
222
223     printf("%s",tostr_rac(s1,&sumaparte));
224     printf("+%s=0\n",tostr_rac(s1,&cx));
225
226     printf("%s=0\n",tostr_rac(s1,&sumacompleta));
227     e->x1=x1;
228     e->x2=x2;
229 }

```



```

230
231 }else if(azero==1){//Lo que se hace si a es igual a cero
232     printf("\n\n%sx^2",tostr_rac(s1,&p));
233     printf("+%sx",tostr_rac(s1,&q));
234     printf("+%s=0\n",tostr_rac(s1,&r));
235
236     mult_rac(&m6,&minusone,&r);
237     divi_rac(&divi4,&m6,&q);
238     printf("\nSolo tien una raiz: %s\n",tostr_rac(s1,&divi4));
239     e->x1=divi4;
240
241 }
242 }

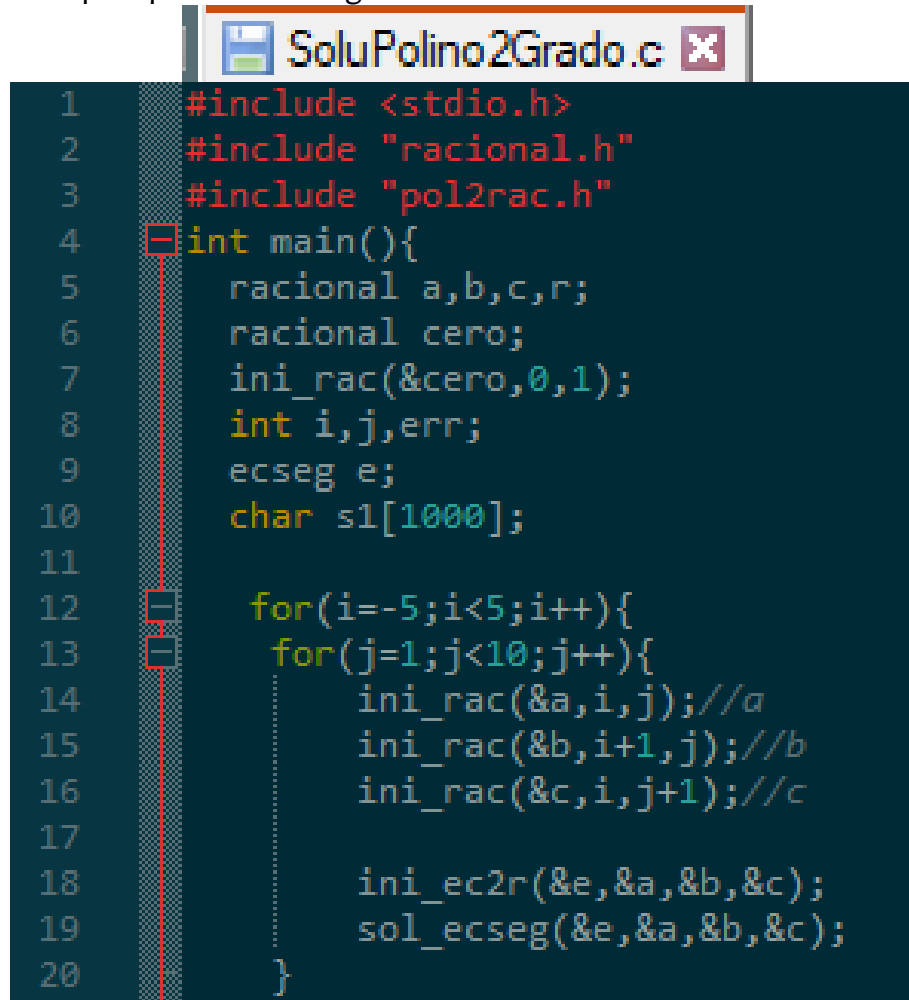
```

CONCLUSIONES

Las pruebas que realice en mi código las corría con el código:

```
gcc SoluPolino2Grado.c pol2rac.c racional.c
```

El archivo main que cree para probar mi código fue este:



```

1  #include <stdio.h>
2  #include "racional.h"
3  #include "pol2rac.h"
4  int main(){
5      racional a,b,c,r;
6      racional cero;
7      ini_rac(&cero,0,1);
8      int i,j,err;
9      ecseg e;
10     char s1[1000];
11
12     for(i=-5;i<5;i++){
13         for(j=1;j<10;j++){
14             ini_rac(&a,i,j);//a
15             ini_rac(&b,i+1,j);//b
16             ini_rac(&c,i,j+1);//c
17
18             ini_ec2r(&e,&a,&b,&c);
19             sol_ecseg(&e,&a,&b,&c);
20         }

```

```

21     }
22     for(i=-5;i<5;i++){
23     for(j=1;j<10;j++){
24         ini_rac(&a,i,j);//a
25         ini_rac(&b,2*i,j);//b
26         ini_rac(&c,i,j+1);//c
27
28         ini_ec2r(&e,&a,&b,&c);
29         sol_ecseg(&e,&a,&b,&c);
30     }
31     }
32     for(i=1;i<10;i++){
33     for(j=1;j<10;j++){
34         ini_rac(&a,i,j);//a
35         ini_rac(&b,i,j);//b
36         ini_rac(&c,i,j+1);//c
37
38         ini_ec2r(&e,&a,&b,&c);
39         sol_ecseg(&e,&a,&b,&c);
40     }
41     }
42     return 0;
43 }
44

```

Cuando corro este código se verifica en todos los casos que tome en cuenta para sacar la raíz :

```

C:\Users\Cesar Hernández\Desktop\AlgoEstrucDat\P11\scr2>gcc SoluPolino2Grado.c pol2rac.c racional.c
C:\Users\Cesar Hernández\Desktop\AlgoEstrucDat\P11\scr2>a

0x^2+(1)x+0=0
Solo tien una raiz: 0

0x^2+(1/2)x+0=0
Solo tien una raiz: 0

0x^2+(1/3)x+0=0
Solo tien una raiz: 0

0x^2+(1/4)x+0=0
Solo tien una raiz: 0

0x^2+(1/5)x+0=0
Solo tien una raiz: 0

```

Cuando el polinomio tiene dos soluciones:

```
(-1/8)x^2+(-1/4)x+(-1/9)=0
El polinomio tiene dos soluciones:
Solucion 1:(-4/3)
Solucion 2:(-2/3)
Comprobacion de soluciones:
siendo x1=(-4/3)
(-1/8)(-4/3)^2+(-1/4)(-4/3)+(-1/9)(-4/3)=0
(-1/8)(16/9)+(1/3)+(4/27)=0
(-2/9)+(1/3)+(4/27)=0
(1/9)+(4/27)=0
(7/27)=0
siendo x2=(-2/3)
(-1/8)(-2/3)^2+(-1/4)(-2/3)+(-1/9)(-2/3)=0
(-1/8)(4/9)+(1/6)+(2/27)=0
(-2/9)+(1/6)+(2/27)=0
(-1/18)+(2/27)=0
(1/54)=0
```

```
(1/7)x^2+(2/7)x+(1/8)=0
El polinomio tiene dos soluciones:
Solucion 1:(-11/18)
Solucion 2:(-25/18)
Comprobacion de soluciones:
siendo x1=(-11/18)
(1/7)(-11/18)^2+(2/7)(-11/18)+(1/8)(-11/18)=0
(1/7)(121/324)+(-11/63)+(-11/144)=0
(121/2268)+(-11/63)+(-11/144)=0
(275/2268)+(-11/144)=0
(407/9072)=0
siendo x2=(-25/18)
(1/7)(-25/18)^2+(2/7)(-25/18)+(1/8)(-25/18)=0
(1/7)(625/324)+(-25/63)+(-25/144)=0
(121/2268)+(-25/63)+(-25/144)=0
(779/2268)+(-25/144)=0
(1541/9072)=0
```

Cuando el polinomio solo tiene una sola solución:

```
(-1/9)x^2+(-2/9)x+(-1/9)=0
El polinomio tiene una sola solucion:
(-1)
Comprobacion de soluciones:
siendo x=(-1)
(-1/9)(-1)^2+(-2/9)(-1)+(-1/9)(-1)=0
(-1/9)(1)+(2/9)+(1/9)=0
(-1/9)+(2/9)+(1/9)=0
(1/9)+(1/9)=0
(2/9)=0
```

Nótese que en algunas respuestas la raíz no es muy exacta pues a la hora de sustituir el resultado en el polinomio original se no siempre da cero, esto pasa más en la situación en donde solo tenemos una solución. Me gustaría recalcar que en el proceso de escribir el código me costo entender que es lo que se tenía que hacer en realidad por lo que no llegue a simplificar la raíz de tal forma que siempre se reduzca lo mas posible y por esa razón no da exacto en las comprobaciones, teniendo en cuenta que una raíz de racionales siempre va a ser inexacta en algún sentido pues es muy difícil que sea exacta tomando racionales.

Para probar esto cree otro main.c en donde hice el código de la resolución de polinomios de segundo grado, pero con flotantes o sea que se calculan las raíces con decimales y en este caso si es mas preciso. Mi intención era tratar de replicar esta situación en la función de los polinomios, pero creo no lo complete al 100. Creo que hice un 90 por ciento del trabajo que se debía hacer y ya solo faltaría encontrar la forma de reducir las raíces delos racionales de forma que los resultados sean mas precisos o en su defecto que se acerquen lo más posible al cero.

Este es el código de mi main con la resolución del polinomio de segundo grado pero con flotantes:

 SolPol2Grad.c 

```
1  #include <stdio.h>
2  #include <math.h>
3  //Idea general para la solucion de polinomios pero con numeros enteros y decimales no contando con los racionales
4  float comprobacion(float a,float b,float c,float sol);
5  int main(){
6      float a,b,c,D;
7      float sol,sol1,sol2;
8      printf("\nEcuaciones de segundo Grado\n");
9      printf("\nax^2 + bx + c = 0\n");
10     printf("Introduce el valor de a:");
11     scanf("%f",&a);
12     printf("Introduce el valor de b:");
13     scanf("%f",&b);
14     printf("Introduce el valor de c:");
15     scanf("%f",&c);
16
17     D=b*b-(4*a*c);
18     if(D==0){
19         sol=-b/(2*a);
20         printf("Solucion: %.2f\n",sol);
21         comprobacion(a,b,c,sol);
22
23     }else if(D>0){
24         sol1=(-b+sqrt(D))/(2*a);
25         sol2=(-b-sqrt(D))/(2*a);
26         printf("Solucion 1: %.2f\n",sol1);
27         printf("Solucion 2: %.2f\n",sol2);
28
29         comprobacion(a,b,c,sol1);
30         comprobacion(a,b,c,sol2);
31
32     }else{ /* D<0 */
33         printf("No tiene solucion.\n");
34         /*
35         printf("Sol. 1: %.2f + %.2f i\n",-b/(2*a),(sqrt(-D))/(2*a));
36         printf("Sol. 2: %.2f - %.2f i\n",-b/(2*a),(sqrt(-D))/(2*a));
37         */
38     }
39 }
```

```

17     D=b*b-(4*a*c);
18     if(D==0){
19         sol=-b/(2*a);
20         printf("Solucion: %.2f\n",sol);
21         comprobacion(a,b,c,sol);
22     }
23     }else if(D>0){
24         sol1=(-b+sqrt(D))/(2*a);
25         sol2=(-b-sqrt(D))/(2*a);
26         printf("Solucion 1: %.2f\n",sol1);
27         printf("Solucion 2: %.2f\n",sol2);
28
29         comprobacion(a,b,c,sol1);
30
31         comprobacion(a,b,c,sol2);
32     }
33     }else{ /* D<0 */
34         printf("No tiene solucion.\n");
35         /*
36         printf("Sol. 1: %.2f + %.2f i\n",-b/(2*a),(sqrt(-D))/(2*a));
37         printf("Sol. 2: %.2f - %.2f i\n",-b/(2*a),(sqrt(-D))/(2*a));
38         */
39     }
40
41     return 0;
42 }
43
44 float comprobacion(float a,float b,float c,float sol){
45     float comprobacion;
46     comprobacion=a*(sol*sol)+b*(sol)+c;
47     comprobacion=fabs(comprobacion); //valor absoluto de la comprobacion
48     printf("\nComprobacion de que la solucion es correcta cuando:\n");
49     printf("x=%.2f\n",sol);
50     printf("ax^2 + bx + c = 0\n");
51     printf("%.2f(%2f)^2 + %2f(%2f) + %2f = %2f\n",a,sol,b,sol,c,comprobacion);
52     printf("%.2f(%2f) + (%2f) + (%2f) = %2f\n",a,sol*sol,b*sol,c,comprobacion);
53     printf("%.2f + (%2f) = %2f\n",a*sol*sol,(b*sol)+c,comprobacion);
54     printf("%.2f=%2f\n",fabs((a*sol*sol)+(b*sol)+c),comprobacion);
55 }

```

Simbolo del sistema

C:\Users\Cesar Hernández\Desktop\AlgoEstrucDat\P11\scr2>a

Ecuaciones de segundo Grado

ax^2 + bx + c = 0

Introduce el valor de a:1.4

Introduce el valor de b:9.5

Introduce el valor de c:4.3

Solucion 1: -0.49

Solucion 2: -6.30

Comprobacion de que la solucion es correcta cuando:

x=-0.49

ax^2 + bx + c = 0

1.40(-0.49)^2 + 9.50(-0.49) + 4.30 = 0.00

1.40(0.24) + (-4.63) + (4.30) = 0.00

0.33 + (-0.33) = 0.00

0.00=0.00

Comprobacion de que la solucion es correcta cuando:

x=-6.30

ax^2 + bx + c = 0

1.40(-6.30)^2 + 9.50(-6.30) + 4.30 = 0.00

1.40(39.67) + (-59.83) + (4.30) = 0.00

55.53 + (-55.53) = 0.00

0.00=0.00

NOTA:Espero y si me cuenta algo el proyecto pues le dedique bastante tiempo para recordar como programar en c y además entender y aprender a usar esta nueva forma de codificar en archivos distintos. Sin mas por el momento esto será todo mi reporte de la practica P11.