

```

1: #include<stdio.h>
2: #include<math.h>
3:
4:
5: void getOffsetToAdd(int a, int b, int c, int x, int *temp)
6: {
7:     int temp3 = c - a * (x - 10);
8:     int val1 = temp3 / b;
9:     temp3 = c - a * (x + 10);
10:    int val2 = temp3 / b;
11:    int x1 = (x - 10) < (x + 10) ? x - 10 : x + 10;
12:    int y1 = val1 < val2 ? val1 : val2;
13:    int offsetToAdd = 0;
14:    if (!(x1 >= 0 && y1 >= 0))
15:        offsetToAdd = x1 < y1 ? x1 : y1;
16:    x1 = (x - 10) > (x + 10) ? x - 10 : x + 10;
17:    y1 = val1 > val2 ? val1 : val2;
18:    offsetToAdd = abs(offsetToAdd);
19:    *temp = x1;
20:    *(temp + 1) = y1;
21:    *(temp + 2) = offsetToAdd;
22: }
23:
24:
25: void plotGraph(int a, int b, int c, int p, int q, int r, int x, int y)
26: {
27:     int temp[3];
28:     getOffsetToAdd(a, b, c, x, temp);
29:     int x1 = temp[0], y1 = temp[1], v1 = temp[2];
30:     getOffsetToAdd(p, q, r, x, temp);
31:     int x2 = temp[0], y2 = temp[1], v2 = temp[2];
32:     int offsetToAdd = v1 < v2 ? v2 : v1;
33:
34:     int x3 = x1 < x2 ? x2 : x1;
35:     int y3 = y1 < y2 ? y2 : y1;
36:     int arr[x3 + offsetToAdd][y3 + offsetToAdd];
37:     int i, j;
38:     for (i = 0; i < x3 + offsetToAdd; i++)
39:     {
40:         for (j = 0; j < y3 + offsetToAdd; j++)
41:             arr[i][j] = 0;
42:     }
43:
44:     for (i = x - 10; i < x + 10; i++)
45:     {
46:         int temp3 = c - a * i;
47:         y = temp3 / b;
48:         arr[i + offsetToAdd][y + offsetToAdd] = 1;
49:     }
50:
51:     for (i = x - 10; i < x + 10; i++)
52:     {
53:         int temp3 = r - p * i;
54:         y = temp3 / q;
55:         arr[i + offsetToAdd][y + offsetToAdd] = 2;
56:     }
57:     printf("\n");
58:     for (i = 0; i < x3 + offsetToAdd; i++)
59:     {
60:         for (j = 0; j < y3 + offsetToAdd; j++)

```

```

61:         arr[i][j] == 0 ? printf("%c ", ' ') : printf("%d ", arr[i][j]);
62:
63:     printf("\n");
64: }
65: }
66:
67:
68: int compare_float(float f1, float f2)
69: {
70:     float precision = 0.00001;
71:     if (fabs(f1 - f2) < precision)
72:         return 1;
73:     return 0;
74: }
75:
76:
77: void substitution(int a, int b, int c, int p, int q, int r, float *result)
78: {
79:     printf("\n\nSolving using Substitution Method:\n\n");
80:     float x = (float)(b*r - q*c)/(p*b - a*q);
81:     printf("The calculated value of x is = %.2f\n", x);
82:     int temp3 = c - a * x;
83:     float y = (float)temp3 / b;
84:     printf("The calculated value of y is = %.2f\n", y);
85:     *result = x;
86:     *(result + 1) = y;
87: }
88:
89:
90: void matrixMultiplication(int a, int b, int c, int p, int q, int r, float *result)
91: {
92:     printf("\n\nSolving using Matrix Multiplication Method: \n\t[X = inverse(A)B]\n\n");
93:     int determinant = a * q - b * p;
94:     printf("The value of determinant = %d", determinant);
95:     printf("\nThe value of resultant A inverse is:\n");
96:     float adj[2][2] = {{q, -b}, {-p, a}};
97:     float inverse[2][2];
98:     int B[2][1] = {{c}, {r}};
99:     int i, j, k;
100:     for (i = 0; i < 2; i++)
101:     {
102:         for (j = 0; j < 2; j++)
103:         {
104:             printf("%.2f\t", (float)adj[i][j] / determinant);
105:             inverse[i][j] = (float)adj[i][j] / determinant;
106:         }
107:         printf("\n");
108:     }
109:     printf("\nThe value of B is:\n");
110:     for (i = 0; i < 2; i++)
111:     {
112:         for (j = 0; j < 1; j++)
113:             printf("%d", B[i][j]);
114:         printf("\n");
115:     }
116:     float res[2][1];
117:     for (i = 0; i < 2; i++)
118:     {
119:         for (j = 0; j < 2; j++)
120:         {

```

```

121:         res[i][j] = 0;
122:         for (k = 0; k < 2; k++)
123:             res[i][j] += inverse[i][k] * B[k][j];
124:     }
125: }
126: printf("The value of resultant matrix is:\n");
127: for (i = 0; i < 2; i++)
128: {
129:     for (j = 0; j < 1; j++)
130:         printf("%.2f", res[i][j]);
131:     printf("\n");
132: }
133: printf("\nThe calculated value of x is = %.2f\n", res[0][0]);
134: printf("The calculated value of y is = %.2f\n", res[1][0]);
135: *result = res[0][0];
136: *(result + 1) = res[1][0];
137: }
138:
139:
140: int main() {
141:     int a, b, c;
142:     int p, q, r;
143:     printf("The equations should be in the format of:\n\tax + by = c and\n\tpx + qy = r\n");
144:     printf("Enter a, b, c, p, q, r: ");
145:     scanf("%d %d %d %d %d %d", &a, &b, &c, &p, &q, &r);
146:
147:     char c1, c2;
148:     c1 = b < 0 ? atoi("45") : atoi("43");
149:     c2 = q < 0 ? atoi("45") : atoi("43");
150:     printf("The provided equations are:\n\t%dx %c %dy = %d\n\t%dx %c %dy = %d", a, c1, abs(b),
151:
152:     float m1 = -(float)a / b;
153:     float m2 = -(float)p / q;
154:     if (compare_float(m1, m2))
155:     {
156:         printf("\n\nThe lines are parallel and solution doesn't exist");
157:         return 0;
158:     }
159:
160:     float s_res[2], m_res[2];
161:     substitution(a, b, c, p, q, r, s_res);
162:     matrixMultiplication(a, b, c, p, q, r, m_res);
163:     if (compare_float(s_res[1], m_res[1]) && compare_float(s_res[0], m_res[0]))
164:     {
165:         printf("\n\nThe solution using both the methods match!");
166:         plotGraph(a, b, c, p, q, r, s_res[0], s_res[1]);
167:     }
168:     else
169:         printf("\n\nThe solution using both methods are different!");
170:     return 0;
171: }

```