

LENGUAJES DE PROGRAMACIÓN

Solución al Trabajo Práctico - Convocatoria extraordinaria de 2026

Ejercicio 1

Escriba un programa en C++ que, dado un punto del plano, calcule su imagen especular respecto a una recta definida por dos puntos.

Los puntos se representan mediante sus coordenadas cartesianas (x, y) .

El programa debe realizar las siguientes acciones:

1. Declarar la estructura siguiente:

```
struct Punto {  
    double x, y;  
};
```

Declarar además dos constantes globales de tipo `Punto`, denominadas `P1` y `P2`, con los valores: $P_1 = (0, 0)$, $P_2 = (1, 0)$.

Estos dos puntos definen la recta respecto a la cual se calculará la simetría.

2. Solicitar al usuario, mediante mensajes en la consola, las coordenadas x e y de un punto P . Almacenar los valores leídos en una variable del tipo `Punto`.
3. Calcular la imagen especular del punto P respecto a la recta definida por los puntos P_1 y P_2 . Llamaremos a esta imagen especular el punto S .
4. Mostrar en la consola las coordenadas cartesianas (x, y) del punto S en formato fijo con cuatro cifras decimales.
5. Finalizar el programa.

Muestre en la memoria capturas de pantalla de los resultados producidos por su programa al ejecutar cada uno de los casos de prueba siguientes.

- **Caso de prueba 1:** $P_1 = (0, 0)$, $P_2 = (1, 0)$, $P = (6, -2)$.
- **Caso de prueba 2:** $P_1 = (-1, 0)$, $P_2 = (0, 0)$, $P = (6, -2)$.
- **Caso de prueba 3:** $P_1 = (0, 0)$, $P_2 = (0, 1)$, $P = (6, -2)$.
- **Caso de prueba 4:** $P_1 = (1, 1)$, $P_2 = (-1, -1)$, $P = (6, -2)$.
- **Caso de prueba 5:** $P_1 = (0.5, -6.2)$, $P_2 = (2.0, 4.5)$, $P = (6, -2)$.

Solución al Ejercicio 1

```
1 #include <iostream>
2 #include <iomanip>
3 #include <cmath>
4
5 struct Punto {
6     double x, y;
7 };
8
9 const Punto P1{0, 0};
10 const Punto P2{1, 0};
11
12 int main() {
13     Punto P;
14
15     std::cout << "Introduzca la coordenada x: ";
16     std::cin >> P.x;
17
18     std::cout << "Introduzca la coordenada y: ";
19     std::cin >> P.y;
20
21     // Vector director unitario de la recta
22     Punto u;
23     u.x = P2.x - P1.x;
24     u.y = P2.y - P1.y;
25
26     double modulo = std::sqrt(u.x * u.x + u.y * u.y);
27
28     u.x /= modulo;
29     u.y /= modulo;
30
31     // Proyeccion ortogonal de P sobre la recta
32     double t = (P.x - P1.x) * u.x + (P.y - P1.y) * u.y;
33
34     Punto H;
35     H.x = P1.x + t * u.x;
36     H.y = P1.y + t * u.y;
37
38     // Imagen especular de P
39     Punto S;
40     S.x = 2 * H.x - P.x;
41     S.y = 2 * H.y - P.y;
42
43     std::cout << std::fixed << std::setprecision(4)
44               << "\nImagen especular:\n"
45               << "(" << S.x << ", " << S.y << ")\n";
46
47     return 0;
48 }
```

Ejercicio 2

Un robot recorre en el plano una trayectoria continua que viene descrita por una secuencia de segmentos rectos. Cada uno de estos segmentos viene definido por su longitud expresada en metros y su ángulo expresado en grados. Este último está definido respecto al semieje positivo del eje X de un sistema de referencia cartesiano XY. El ángulo no solo indica la inclinación del segmento, sino también el sentido en el cual el robot lo recorre.

El ángulo está expresado en la determinación $[0^\circ, 360^\circ)$, indicando el giro en el sentido antihorario. Por ejemplo, un ángulo de 0° corresponde al semieje X positivo, 90° al semieje Y positivo, 180° al semieje X negativo y 270° al semieje Y negativo.

El robot recorre los segmentos a una velocidad de módulo constante V , expresada en metros por segundo. Cuando el robot alcanza el punto de unión de dos segmentos, se detiene, realiza el giro, y a continuación reanuda la marcha. El tiempo que emplea el robot en hacer el giro es T segundos. Se supone que el frenado y acelerado son instantáneos. Durante el tiempo empleado en el giro, la posición del robot permanece constante.

El robot se encuentra inicialmente en el punto de coordenadas (x_0, y_0) . En el instante inicial, el robot comienza a recorrer el primer segmento. Cuando llega al final del último segmento, el robot se detiene, permaneciendo en la posición final.

En un fichero de texto llamado *trayectoria.txt* está almacenada, en dos columnas separadas por uno o más espacios en blanco, la información de la secuencia ordenada de segmentos de la trayectoria:

- En la primera columna está la longitud del segmento expresada en metros. Es un número positivo mayor que cero.
- En la segunda columna está el ángulo del segmento respecto al semieje X positivo del sistema de referencia cartesiano XY.

A continuación se muestran las primeras líneas del fichero. Obsérvese que la primera línea contiene una cabecera que indica el significado de cada columna.

```
longitud(m)  angulo(grados)
4.33 16.46
6.81 288.59
4.86 134.90
```

Escriba un programa en C++ que calcule y escriba en la consola, con dos dígitos decimales, las coordenadas de la posición (x, y) del robot cuando han transcurrido t segundos desde el instante inicial.

Para ello, el programa debe primeramente solicitar al usuario, mediante un mensaje escrito en la consola, que introduzca por teclado los datos siguientes:

- Las coordenadas de la posición inicial (x_0, y_0) expresadas en metros.
- La velocidad V expresada en metros por segundo.
- El tiempo T expresado en segundos.
- El tiempo t expresado en segundos.

El programa debe leer y almacenar los valores introducidos por el usuario, comprobando que $V > 0$, $T \geq 0$ y $t \geq 0$. Si no se cumple alguna de estas condiciones, el programa debe terminar. A continuación el programa debe, a partir de la información contenida en el fichero *trayectoria.txt*, calcular y escribir en la consola la posición (x, y) del robot cuando han transcurrido t segundos desde el instante inicial, y finalmente terminar.

Puede asumir que el fichero no contiene errores de formato. Si se produce error al abrir el fichero, el programa debe terminar.

Debajo de estas líneas se muestra el contenido del fichero *trayectoria.txt* que debe emplear para ejecutar los casos de prueba descritos a continuación. Incluya en la memoria capturas de pantalla que muestren el resultado obtenido al ejecutar su programa para cada uno de los casos de prueba indicados.

```
longitud(m)  angulo(grados)
4.33  16.46
6.81  288.59
4.86  134.90
5.87  81.32
1.65  254.95
6.28  50.65
4.29  32.13
2.28  100.23
7.48  231.29
3.16  70.65
2.20  301.31
9.62  337.31
2.14  320.04
5.51  10.95
4.02  170.16
```

3.57 33.01
 2.82 36.62
 8.37 10.33
 1.92 299.26
 8.54 75.16

CASOS DE PRUEBA

	$(x_0 \text{ [m]}, y_0 \text{ [m]})$	$V \text{ [m/s]}$	$T \text{ [s]}$	$t \text{ [s]}$
Caso de prueba 1:	$(6.82, -10.32)$	2.00	2.00	0
Caso de prueba 2:	$(-0.54, -60.00)$	1.50	2.00	10000.00
Caso de prueba 3:	$(0.50, -62.00)$	2.00	0.50	6.27
Caso de prueba 4:	$(3.87, -11.40)$	1.16	0.35	75.81
Caso de prueba 5:	$(-3.87, 11.40)$	12.35	0.00	120.00

Solución al Ejercicio 2

```
1 #include <cmath>
2 #include <complex>
3 #include <fstream>
4 #include <iomanip>
5 #include <iostream>
6 #include <string>
7
8 const std::string NOMBRE_FICHERO = "trayectoria.txt";
9 const double PI = acos(-1.0);
10
11 struct Segmento {
12     double longitud, angulo;
13 };
14
15 int main() {
16     // Solicitar los datos al usuario
17     double x0, y0, V, T, t;
18
19     std::cout << "x0 = ";
20     std::cin >> x0;
21
22     std::cout << "y0 = ";
23     std::cin >> y0;
24
25     std::cout << "V = ";
26     std::cin >> V;
27
28     std::cout << "T = ";
29     std::cin >> T;
30
31     std::cout << "t = ";
32     std::cin >> t;
33
34     // Comprobar la validez de los datos
35     if (V <= 0.0 || T < 0.0 || t < 0.0) {
36         std::cerr << "Datos no validos\n";
37         return 1;
38     }
39
40     // Abrir el fichero
41     std::ifstream f(NOMBRE_FICHERO.c_str(), std::ios::in);
42     if (!f) {
43         std::cerr << "Error al abrir el fichero\n";
44         return 1;
45     }
46
47     // Leer y descartar la cabecera
48     std::string linea;
49     std::getline(f, linea);
50
51     // Posicion actual del robot
```

```

52     std::complex<double> z(x0, y0);
53
54     // Tiempo transcurrido desde el instante inicial
55     double tiempo = 0.0;
56
57     bool encontrado = false;
58     Segmento s;
59
60     while (!encontrado && f >> s.longitud >> s.angulo) {
61         double rad = s.angulo * PI / 180.0;
62         double dt = s.longitud / V;
63
64         // Comprobar si t cae durante el recorrido de este segmento
65         if (t <= tiempo + dt) {
66             z += std::polar(V * (t - tiempo), rad);
67             encontrado = true;
68         } else {
69             // Avanzar hasta el final del segmento.
70             z += std::polar(s.longitud, rad);
71             tiempo += dt;
72
73             // Comprobar si t cae durante el giro.
74             if (t <= tiempo + T)
75                 encontrado = true;
76             else
77                 tiempo += T;
78         }
79     }
80
81     // Si no se ha encontrado la posicion antes,
82     // el robot ya ha terminado la trayectoria y
83     // permanece detenido en la posicion final.
84
85     std::cout << std::fixed << std::setprecision(2)
86               << "(" << z.real() << ", " << z.imag() << ") \n";
87 }
    
```

Ejercicio 3

Consideremos un examen que consta de 20 preguntas de tipo test. En un fichero de texto llamado `respuestas.txt` se almacenan las respuestas de los alumnos. Cada línea del fichero contiene los tres siguientes campos separados por uno o más espacios en blanco:

- Identificador del alumno (cadena de caracteres).
- Número de la pregunta (entero comprendido entre 1 y 20).
- Respuesta del alumno (carácter 'A', 'B', 'C' o 'D') o bien el carácter '-' para indicar una pregunta no contestada.

Por ejemplo, las primeras líneas del fichero son:

```
Alumno1 4 A
Alumno1 2 B
Alumno5 10 C
Alumno1 3 -
Alumno2 1 C
```

No asuma que las respuestas de un mismo alumno están escritas consecutivamente en el fichero, ni tampoco que están ordenadas por número de pregunta.

Para cada alumno existe exactamente un registro correspondiente a cada una de las 20 preguntas del examen.

El número de alumnos es desconocido a priori y se determinará al leer el fichero.

Puede suponer que el fichero no contiene información duplicada ni contradictoria, ni tampoco errores de formato.

Escriba un programa en C++ que realice las acciones siguientes:

1. Declarar un array global de 20 componentes tipo *char* llamado `aSolucion` e inicializarlo con las respuestas correctas del examen, que son las siguientes:

```
1-C,    2-A,    3-D,    4-B,    5-A,
6-C,    7-B,    8-D,    9-C,    10-A,
11-B,   12-D,   13-A,   14-C,   15-D,
16-B,   17-C,   18-A,   19-B,   20-D
```

2. Leer el fichero, guardando los datos de cada alumno en una variable estructura de tipo `DatosAlumno`, la cual es declarada de la forma siguiente:

```
struct DatosAlumno {
    int aciertos;
    int fallos;
    int blancos;
};
```

Si se produce error al abrir el fichero, escribir un mensaje en la consola indicándolo y terminar.

3. Calcular la nota obtenida por cada alumno en el test, aplicando la fórmula siguiente:

$$\text{nota} = \text{aciertos} \times 0.5 - \text{fallos} \times 0.25$$

Obsérvese que las respuestas no contestadas ni suman, ni restan.

Si el resultado obtenido es negativo, se asignará una nota de 0 puntos.

4. Escribir en la consola la información siguiente de cada uno de los alumnos, con un alumno por línea:

- Identificador del alumno.
- Porcentajes de respuestas correctas, falladas y no contestadas sobre el total de preguntas. Los porcentajes se escribirán en formato fijo con cero dígitos decimales y seguidos del carácter '% '.
- Nota en el test escrita en formato fijo con dos dígitos decimales.

La información debe escribirse por orden alfabético del identificador.

5. Finalizar el programa.

Incluya en la memoria una captura de pantalla que muestre el resultado obtenido al ejecutar el siguiente caso de prueba empleando su programa.

CASO DE PRUEBA

Muestre el resultado de la ejecución de su programa para el fichero *respuestas.txt* mostrado a continuación.

Alumno1 10 A
Alumno5 5 A
Alumno5 16 C
Alumno6 15 C
Alumno5 13 B
Alumno2 11 B
Alumno5 14 -
Alumno5 8 A
Alumno5 17 D
Alumno6 10 C
Alumno3 8 D
Alumno4 20 D
Alumno5 9 B
Alumno2 13 A
Alumno1 17 -
Alumno4 3 D
Alumno5 3 C
Alumno5 18 B
Alumno3 12 C
Alumno4 9 C
Alumno1 9 C
Alumno4 2 -
Alumno5 19 C
Alumno6 9 D
Alumno1 8 D
Alumno1 2 A
Alumno4 19 A
Alumno4 11 B
Alumno2 7 B
Alumno5 1 A
Alumno2 2 A
Alumno3 13 A
Alumno3 2 B
Alumno3 11 B
Alumno4 7 A
Alumno2 20 D
Alumno4 8 D
Alumno2 15 A
Alumno4 1 C
Alumno2 19 B
Alumno3 20 D
Alumno3 16 A
Alumno4 17 C
Alumno2 3 D
Alumno4 4 B
Alumno4 6 C
Alumno2 4 B
Alumno3 1 C
Alumno5 20 A
Alumno6 13 D
Alumno2 5 A
Alumno1 7 B
Alumno3 17 C
Alumno5 11 D
Alumno1 19 -
Alumno3 10 A
Alumno6 19 D
Alumno3 3 D
Alumno1 3 D
Alumno6 4 A
Alumno4 13 A

LENGUAJES DE PROGRAMACIÓN

Alumno4 14 D
Alumno2 17 C
Alumno3 7 -
Alumno2 18 A
Alumno1 11 B
Alumno4 15 D
Alumno1 16 B
Alumno3 19 D
Alumno1 6 C
Alumno2 14 -
Alumno3 5 A
Alumno3 6 C
Alumno1 13 A
Alumno3 9 A
Alumno6 2 D
Alumno6 11 A
Alumno6 20 A
Alumno6 1 -
Alumno1 20 D
Alumno5 6 -
Alumno3 4 A
Alumno6 5 C
Alumno2 1 C
Alumno1 1 C
Alumno6 16 D
Alumno6 6 D
Alumno3 18 A
Alumno6 14 A
Alumno3 14 -
Alumno6 8 -
Alumno5 10 C
Alumno5 4 D
Alumno5 12 A
Alumno2 6 -
Alumno4 12 -
Alumno6 3 B
Alumno4 18 A
Alumno4 5 B
Alumno2 10 A
Alumno2 8 D
Alumno6 7 A
Alumno6 18 B
Alumno1 5 A
Alumno3 15 D
Alumno4 16 C
Alumno1 12 D
Alumno4 10 B
Alumno6 12 B
Alumno5 7 D
Alumno1 14 C
Alumno6 17 A
Alumno1 18 A
Alumno2 9 C
Alumno2 12 D
Alumno2 16 B
Alumno5 15 A
Alumno1 4 B
Alumno1 15 D
Alumno5 2 B

Solución al Ejercicio 3

```

1  #include <fstream>
2  #include <iomanip>
3  #include <iostream>
4  #include <map>
5  #include <string>
6  #include <algorithm>
7
8  // Nombre del fichero de entrada
9  const char nombreFich[] = "respuestas.txt";
10
11 // Respuestas correctas del examen
12 const char aSolucion[20] = {
13     'C', 'A', 'D', 'B', 'A',
14     'C', 'B', 'D', 'C', 'A',
15     'B', 'D', 'A', 'C', 'D',
16     'B', 'C', 'A', 'B', 'D'
17 };
18
19 // Informacion asociada a cada alumno
20 struct DatosAlumno {
21     int aciertos;
22     int fallos;
23     int blancos;
24 };
25
26 int main() {
27     // Abrir el fichero.
28     std::ifstream f(nombreFich, std::ios::in);
29     if (!f) {
30         std::cerr << "Error al abrir el fichero " << nombreFich << ".\n";
31         return 1;
32     }
33
34     // Mapa cuya clave es el identificador del alumno
35     // y cuyo valor contiene sus contadores
36     std::map<std::string, DatosAlumno> alumnos;
37
38     std::string id;
39     int pregunta;
40     char respuesta;
41
42     // Leer todas las respuestas del fichero
43     while (f >> id >> pregunta >> respuesta) {
44         // Si el alumno aparece por primera vez,
45         // inicializar sus contadores
46         if (alumnos.find(id) == alumnos.end())
47             alumnos[id] = {0, 0, 0};
48
49         // Actualizar los contadores
50         if (respuesta == '-')
51             alumnos[id].blancos++;
52         else if (respuesta == aSolucion[pregunta - 1])

```

```

53         alumnos[id].aciertos++;
54     else
55         alumnos[id].fallos++;
56 }
57
58 // Escribir los resultados en la consola
59 for (std::map<std::string, DatosAlumno>::const_iterator it =
60     alumnos.begin(); it != alumnos.end(); ++it) {
61
62     double pAciertos = 100.0 * it->second.aciertos / 20.0;
63     double pFallos    = 100.0 * it->second.fallos / 20.0;
64     double pBlancos   = 100.0 * it->second.blancos / 20.0;
65     double nota = std::max(
66         0.5 * it->second.aciertos - 0.25 * it->second.fallos,
67         0.0);
68
69     std::cout << it->first
70         << std::fixed << std::setprecision(0)
71         << ", Correctas: "      << pAciertos << '%'
72         << ", Falladas: "      << pFallos    << '%'
73         << ", No contestadas: " << pBlancos  << '%'
74         << std::fixed << std::setprecision(2)
75         << " Nota: " << nota << std::endl;
76 }
77
78 return 0;
79 }

```

Ejercicio 4

Consideremos los ejes cartesianos X - Y en el plano. Una inecuación del tipo

$$a \cdot x + b \cdot y > c$$

divide el plano en dos semiplanos: uno factible, en el cual los puntos satisfacen la inecuación, y otro no factible, donde no la satisfacen.

Si se escriben varias inecuaciones que deben ser satisfechas simultáneamente, se define una **región factible**, que puede ser vacía.

Escriba un programa en C++ que, dado un conjunto de inecuaciones introducido por el usuario a través de la consola y dado un punto del plano, determine si dicho punto pertenece a la región factible definida por todas las inecuaciones.

El programa no debe calcular explícitamente la región factible, ni determinar si esta es vacía. Únicamente debe comprobar si el punto introducido por el usuario satisface todas las inecuaciones.

El programa debe realizar las siguientes acciones:

1. Escribir en la consola un mensaje preguntando al usuario cuántas inecuaciones tiene el problema. Leer el valor introducido por el usuario y almacenarlo en una variable entera llamada `nIneq`.
2. Comprobar que `nIneq` es mayor que cero. En caso contrario, mostrar un mensaje indicándolo y terminar el programa.
3. Leer de la consola `nIneq` líneas de texto. Antes de validar cada línea, deben ignorarse todos los espacios en blanco que contenga. Una vez hecho esto, una inecuación válida debe tener una de las formas siguientes:

$$a * x + b * y > c$$

$$a * x - b * y > c$$

donde:

- a y c son números enteros o reales en formato fijo, con signo opcional.
- b es un número entero o real en formato fijo sin signo.
- El signo del término asociado a y viene determinado exclusivamente por el símbolo '+' o '-' que separa ambos términos.

- Los símbolos '*', '+', '-', 'x', 'y' y '>' deben aparecer exactamente en las posiciones indicadas.

Si alguna de las líneas leídas no cumple este formato, deberá mostrarse un mensaje de error, descartarse la línea y solicitarse nuevamente al usuario.

- Mediante un mensaje en la consola, solicitar al usuario que introduzca las coordenadas x e y de un punto del plano. Leer ambos valores y almacenarlos en una variable estructura llamada P , con dos campos de tipo `double` llamados x e y .
- Determinar si el punto almacenado en P satisface simultáneamente todas las inecuaciones válidas introducidas por el usuario. Escribir en la consola uno de los mensajes siguientes:
 - El punto pertenece a la región factible
 - El punto no pertenece a la región factible
- Terminar el programa.

Muestre en la memoria las capturas de pantalla obtenidas al ejecutar los siguientes casos de prueba.

Caso de prueba 1

Numero de inecuaciones: 1

Inecuación:
 $2x + 3y > 6$

Punto:
 $x = 2$
 $y = 1$

Caso de prueba 2

Numero de inecuaciones: 2

Inecuaciones:
 $1.5x + 2.0y > 4.0$
 $-0.5x + 1.2y > -1.0$

Punto:
 $x = 2.5$
 $y = 1.5$

Caso de prueba 3

Numero de inecuaciones: 2

Inecuaciones:

$$1.5*x+2.0*y>4.0$$

$$-0.5*x+1.2*y>-1.0$$

Punto:

$$x = 0.5$$

$$y = 1.0$$

Caso de prueba 4

Numero de inecuaciones: 10

Inecuaciones:

$$1.2*x+0.5*y>2.0$$

$$-0.8*x+1.5*y>-1.0$$

$$2.0*x-0.3*y>1.0$$

$$0.7*x+1.1*y>3.0$$

$$-1.0*x+2.0*y>0.0$$

$$1.5*x-0.5*y>2.0$$

$$0.4*x+0.9*y>1.5$$

$$-0.2*x+1.3*y>2.0$$

$$1.8*x+0.2*y>4.0$$

$$-1.1*x+1.6*y>1.0$$

Punto:

$$x = 3.0$$

$$y = 3.0$$

Caso de prueba 5

Compruebe que su programa detecta los errores de formato de las inecuaciones siguientes:

$$3x+2*y>5$$

$$3*x+2y>5$$

$$3*x+2*y>=5$$

$$3*x+2*y<5$$

$$3*x+2*y=5$$

$$3**x+2*y>5$$

$$3*x++2*y>5$$

$$3*x+-2*y>5$$

$$3*x+*y>5$$

$$*x+2*y>5$$

$$3*x+2*>5$$

$$3*x+2*y>$$

Solución al Ejercicio 4

```

1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cstdlib>
5  #include <limits>
6
7  struct Punto {
8      double x;
9      double y;
10 };
11
12 struct Inecuacion {
13     double a;
14     double b;
15     double c;
16 };
17
18 std::string eliminaEspacios(const std::string &s) {
19     std::string resultado;
20
21     for (unsigned int i = 0; i < s.length(); i++)
22         if (s[i] != ' ')
23             resultado += s[i];
24
25     return resultado;
26 }
27
28 bool esDigito(char c) {
29     return c >= '0' && c <= '9';
30 }
31
32 bool leeNumero(const std::string &s, unsigned int &i,
33               bool signoPermitido) {
34     if (signoPermitido)
35         if (i < s.length() && (s[i] == '+' || s[i] == '-'))
36             i++;
37
38     bool hayDigitos = false;
39
40     while (i < s.length() && esDigito(s[i])) {
41         hayDigitos = true;
42         i++;
43     }
44
45     if (i < s.length() && s[i] == '.') {
46         i++;
47
48         while (i < s.length() && esDigito(s[i])) {
49             hayDigitos = true;
50             i++;
51         }

```

```

52     }
53
54     return hayDigitos;
55 }
56
57 bool formatoCorrecto(const std::string &texto) {
58     std::string s = eliminaEspacios(texto);
59     unsigned int i = 0;
60
61     if (!leeNumero(s, i, true))
62         return false;
63
64     if (i >= s.length() || s[i] != '*')
65         return false;
66
67     i++;
68
69     if (i >= s.length() || s[i] != 'x')
70         return false;
71
72     i++;
73
74     if (i >= s.length() || (s[i] != '+' && s[i] != '-'))
75         return false;
76
77     i++;
78
79     if (!leeNumero(s, i, false))
80         return false;
81
82     if (i >= s.length() || s[i] != '*')
83         return false;
84
85     i++;
86
87     if (i >= s.length() || s[i] != 'y')
88         return false;
89
90     i++;
91
92     if (i >= s.length() || s[i] != '>')
93         return false;
94
95     i++;
96
97     if (!leeNumero(s, i, true))
98         return false;
99
100    return i == s.length();
101 }
102
103 Inecuacion leerInecuacion(const std::string &texto) {
104     std::string s = eliminaEspacios(texto);
105

```

```

106     unsigned int posX = s.find("*x");
107     unsigned int posY = s.find("*y");
108     unsigned int posMayor = s.find(">");
109
110     unsigned int posSigno = posX + 2;
111
112     std::string aStr = s.substr(0, posX);
113     char signo = s[posSigno];
114     std::string bStr = s.substr(posSigno + 1,
115                                posY - posSigno - 1);
116     std::string cStr = s.substr(posMayor + 1);
117
118     Inecuacion ineq;
119
120     ineq.a = std::atof(aStr.c_str());
121     ineq.b = std::atof(bStr.c_str());
122     ineq.c = std::atof(cStr.c_str());
123
124     if (signo == '-')
125         ineq.b = -ineq.b;
126
127     return ineq;
128 }
129
130 bool satisface(const Inecuacion &ineq, const Punto &p) {
131     return ineq.a * p.x + ineq.b * p.y > ineq.c;
132 }
133
134 bool perteneceRegionFactible(
135     const std::vector<Inecuacion> &inecuaciones,
136     const Punto &p) {
137     for (unsigned int i = 0; i < inecuaciones.size(); i++)
138         if (!satisface(inecuaciones[i], p))
139             return false;
140
141     return true;
142 }
143
144 int main() {
145     int nIneq;
146
147     std::cout << "Numero de inecuaciones: ";
148     std::cin >> nIneq;
149
150     if (nIneq <= 0) {
151         std::cerr << "El numero de inecuaciones debe ser > 0."
152                 << std::endl;
153         return 1;
154     }
155
156     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
157
158     std::vector<Inecuacion> inecuaciones;
159

```

```
160     for (int i = 0; i < nIneq;) {
161         std::string linea;
162
163         std::cout << "Inecuacion " << i + 1 << ": ";
164         std::getline(std::cin, linea);
165
166         if (formatoCorrecto(linea)) {
167             inecuaciones.push_back(LeerInecuacion(linea));
168             i++;
169         } else
170             std::cout << "Formato incorrecto. Vuelva a introducirla."
171                         << std::endl;
172     }
173
174     Punto P;
175
176     std::cout << "Coordenada x: ";
177     std::cin >> P.x;
178
179     std::cout << "Coordenada y: ";
180     std::cin >> P.y;
181
182     if (perteneceRegionFactible(inecuaciones, P))
183         std::cout << "El punto pertenece a la region factible."
184                     << std::endl;
185     else
186         std::cout << "El punto no pertenece a la region factible."
187                     << std::endl;
188
189     return 0;
190 }
```