

Practica 2

Ejercicio 1

Programar una aplicación en C++ que permita manejar curva en el plano formada por una serie de segmentos rectos. Para ello habrá que manejar los siguientes tipos de objetos: Punto, Vector y Curva

Punto

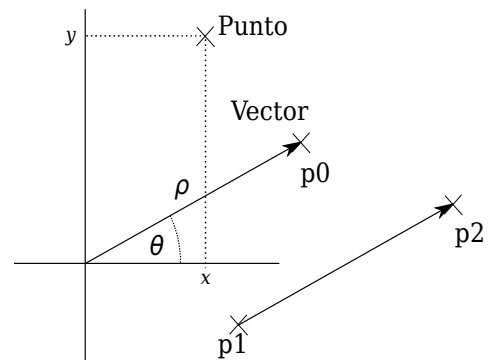
Representa un punto en el plano.

La clase debe tener los siguientes métodos:

- `Punto(double x, double y)` construye punto con las coordenadas dadas.
- `double getX()` devuelve la coordenada x.
- `double getY()` devuelve la coordenada y.
- `string toString()` devuelve `std::string` como “p(x, y)” con las coordenadas.

También:

- sobrecargar el operador `<<` para que se vuelque por un `std::ostream` lo mismo que `toString()`.



Vector

Representa un **vector libre** en el plano. Puede quedar definido: por su módulo y su ángulo; a través de su punto final, considerando el vector que va del punto (0,0) a dicho punto; o a través de dos puntos, como el vector que va del primero al segundo.

La clase deberá tener los siguientes métodos:

- `Vector(double modulo, double angulo)` constructor con módulo y ángulo.
- `Vector(Punto p0)` constructor con punto final.
- `Vector(Punto p1, Punto p2)` vector que va de p1 a p2.
- `double getModulo()` devuelve el módulo.
- `double getAngulo()` devuelve ángulo.
- `Punto getPtoFinal()` devuelve el punto final.
- `std::string toString()` devuelve string “v(modulo, ángulo)”.
- `std::string toStringPt()` devuelve string “vp(x, y)” con las coordenadas del punto final.

También:

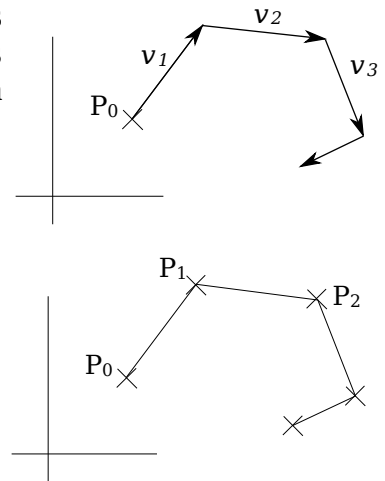
- sobrecargar el operador `<<` para que se vuelque por un `std::ostream` lo mismo que `toString()`.

Curva

Representa una curva en el plano formada por una serie de segmentos rectos. Se puede ver como un punto inicial y una serie de vectores puestos uno a continuación del otro, o como un conjunto de puntos en orden.

La clase debe tener los siguientes métodos:

- `Curva(Punto pto)` construye curva vacía con su punto inicial.
- `void aniaade(Punto pto)` añade punto al final de la curva.
- `void aniaade(Vector vec)` añade vector al final de la curva.
- `unsigned getNumTramos()` devuelve el número de tramos.
- `Punto getPtoInicial()` devuelve el punto inicial.
- `Punto getPtoI(int i)` devuelve punto i-ésimo, siendo 0 el primero.
- `Vector getVector(int i)` devuelve el vector i-ésimo, siendo 0 el primero.
- `void quitaUltimo()` elimina el último tramo.
- `bool esCerrada()` si es una curva cerrada (ultimo punto coincide con el punto inicial).
- `double largo()` longitud total de la curva.
- `std::string toStringPt()` devuelve string “c[p(x0,y0), p(x1,y1) ...]” con los puntos de la curva.
- `std::string toStringVec()` devuelve strig “c[p(x0,y0), vp(x1, y1) ...]” con el punto inicial y los vectores que unen los sucesivos puntos.



También:

- sobrecargar el operador `<<` para que se vuelque por un `std::ostream` lo mismo que `toStringPt()`.

Importante

- ➔ Para la presentación (`toString()` y salida por `ostream`) de los datos `double` se mostrará en punto fijo y con un decimal.
- ➔ Los ángulos recibidos y devueltos por los métodos (y mostrados por `cout`) se considerarán en grados, usando preferiblemente el rango -180° a 180° . Tener presente que las funciones trigonométricas de la librería `<cmath>` trabajan en radianes.

Ejercicio 2

Programar una aplicación en C++ que permita simular circuitos con puertas lógicas.

Los tipos de puertas a implementar son:

- Puertas And de dos entradas cuya salida será el resultado de la y-lógica de sus entradas.
- Puertas Or de dos entradas y cuya salida será el resultado de la o-lógica de sus entradas.
- Puertas Not de una entrada y cuya salida será no-lógica de sus entradas.
- Puertas Uno sin entrada y cuya salida es siempre 1.
- Puertas Cero sin entrada y cuya salida es siempre 0.

Cada puerta estará identificada por un número entero (identificador) distinto del de las demás puertas. Los identificadores comenzarán en 0 y se incrementarán de 1 en 1, es decir, la primera puerta creada debe tener el identificador 0, la segunda 1, la tercera 2, y así sucesivamente.

Los estados posibles para una salida serán: 1, 0 ó ND (no determinado = 9) cuando alguna de las entradas de la puerta no esté conectada o el valor recibido sea, a su vez, ND.

Cada puerta tendrá los siguientes métodos:

- `int salida()` devolverá en valor correspondiente según los valores de entrada, salvo Uno y Cero que devolverán siempre 1 y 0 respectivamente.
- `std::string info(int numEspacios=0)` devuelve un `std::string` con: el tipo, el identificador y el valor de salida de la puerta. Todo ello en una línea precedida de `numEspacios` espacios. En la siguientes líneas aparecerá la información de las puertas que están conectadas en cada una de sus entradas, precedidas, en este caso, por dos espacios más.

Dependiendo del número de entradas que disponga podrá tener también:

- `void conecta1(Puerta* pt)` recibirá un puntero a un objeto de alguno de los tipos de puerta y con ello se definirá la puerta que está conectada a la primera entrada.
- `void conecta2(Puerta* pt)` ídem que el anterior para la segunda entrada (si la hubiera).

Para simular un circuito se crearán y conectarán las puertas necesarias y posteriormente se invocará el método `salida()` de aquellas puertas que sean salida del sistema completo.

Para conocer la estructura del circuito conectado se invocará el método `info()` de aquellas puertas que sean salida del sistema completo.

Ejemplo

Para el siguiente circuito el código de ejecución será:

```
int main() {
    And a0; // id 0
    And a1; // id 1
    Or o2; // id 2
    Not n3; // id 3
    Uno u4; // id 4
    Cero c5; // id 5

    a0.conecta1(&u4);
    a0.conecta2(&u4);

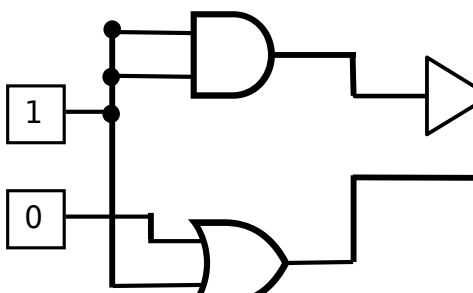
    n3.conecta1(&a0);

    a1.conecta1(&n3);
    a1.conecta2(&o2);

    o2.conecta1(&c5);
    o2.conecta2(&u4);

    std::cout << "La estructura del circuito es :\n"
                << a1.info() << std::endl;

    std::cout << "La salida del circuito es " << a1.salida() ;
    if ( a1.salida() == 0 )
        std::cout << " Correcta";
    else
        std::cout << " Incorrecta";
    std::cout << std::endl;
}
```



La salida será algo como:

```
La estructura del circuito es :
Puerta AND (id=1) salida = 0
Puerta NOT (id=3) salida = 0
Puerta AND (id=0) salida = 1
Puerta UNO (id=4)
Puerta UNO (id=4)
Puerta AND (id=2) salida = 1
Puerta CERO (id=5)
Puerta UNO (id=4)

La salida del circuito es 0 Correcta
```

También

1. Definir una nueva puerta Fijo, similar a Uno y Cero, pero a la que se puede asignar el valor 1 o 0 con los métodos `pon1()` y `pon0()`.
2. Realizar las modificaciones para que se le pueda dar a las puertas un nombre en la construcción `Puerta(std::string nom)`. En ese el caso, se usará también dicho nombre al sacar los datos en `info()`.
3. Definir puertas `AndM` y `OrM` puedan tener un número variable de entradas. Tendrán método `conecta(Puerta* pt)` que añadirá una entrada a la puerta y hará la conexión. Invocada como `conecta(Puerta* pt, int i)` conectará `pt` a la entrada `i`-ésima, sustituyendo a la que existiera. Si en ese momento la puerta tiene `j < (i - 1)` entradas, las entradas intermedias

(de la $j+1$ a la $i-1$) quedarán como no conectadas.

Guía de estilo

Al escribir el código fuente se debe de respetar las siguientes reglas:

- Indicar autor y fecha en la primera línea del fichero fuente.
- Usar dos espacios para cada nivel de sangrado, utilizando caracteres espacio.
- En cada línea no deberá haber más de una instrucción; aunque una instrucción puede ocupar varias líneas. En ese caso las líneas de continuación deberán tener dos niveles de sagrado más que la inicial (4 espacios más).
- No utilizar `using namespace std` sino especificar el espacio de nombre cuando sea necesario (`std::cout`, `std::endl`, `std::string`, `std::vector`, etc.)
- Dejar un espacio antes y después de cada operador, en especial de los de asignación (`=`, `+=`, `-=`, ...), comparación (`>`, `<`, `==`, ...) y los de envío y recepción de `stream` (`<<` y `>>`).
- Las líneas deberán tener, preferiblemente, menos de 72 caracteres y nunca superar los 80.
- Utilizar mensajes de depuración allí donde sea conveniente. Se utilizará la salida de error (`std::cerr`) para mostrar dichos mensajes.
- Se usarán dos ficheros para cada clase: en `Clase.hpp` estará la declaración, y en `Clase.cpp` la definición de TODOS los métodos (no usar métodos *inline*).
- Los atributos de una clase, o bien tendrán un identificador que comience por el carácter barra baja (`_atributo`), o bien se accederán siempre a través del puntero `this` (`this->atributo`).

Guía de diseño

- Si la información se puede representar de varias maneras equivalentes, almacenar en el objeto **sólo una** de las representaciones.
- Optimizar el código, es decir, tratar de utilizar los métodos ya definidos para que no aparezca código que realice la misma funcionalidad en dos sitios distintos.
- Poner atributo `const` a los métodos que no modifican el objeto implicado.
- Realizar una jerarquía de clases donde se utilice la mayor abstracción posible.